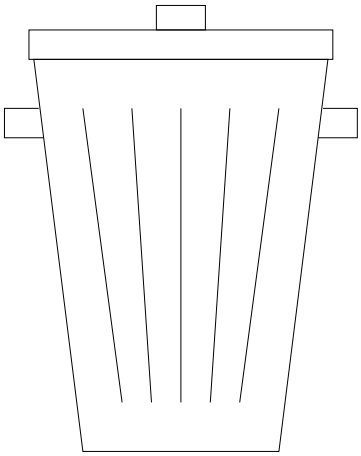
Capitoli scartati

I capitoli di questo volume sono stati scartati durante lo sviluppo dell'opera. Si tratta di materiale potenzialmente errato, superato, incompleto o non curato a sufficienza.



1

ISBN 978-88-905012-2-7

Via Morganella Est, 21 -- I-31050 Ponzano Veneto (TV) -- appunti2@gmail.com

L'opera, nei file sorgenti e nella composizione finale, include degli esempi in forma di sequenze animate, contenenti eventualmente anche delle spiegazioni vocali. Si tratta di video brevi e di qualità molto bassa. Tuttavia, a seconda di come viene diffusa o fruita l'opera, può darsi che sia necessario assolvere a degli obblighi di legge.

Il numero ISBN 978-88-905012-2-7 si riferisce all'opera originale in formato elettronico, pubblicata a titolo gratuito.

Se quest'opera viene consultata in-linea, attraverso uno spazio web, le informazioni generali sull'accesso ai file dell'opera (indirizzo IP di origine, nome del provider della connessione di origine, nome e versione del navigatore usato per accedere, geolocalizzazione dell'origine, data e orario di accesso), assieme al dettaglio delle pagine visitate o dei file scaricati, potrebbero essere annotate in un registro (log) di tale spazio web. Ciò per fini di controllo e statistica, a qualunque titolo. La conservazione di tale registro, se presente, dipende dalla politica del gestore e potrebbe avvenire per una durata di tempo indeterminata.

Quest'opera non contiene tecnologie atte a raccogliere e annotare i dati personali degli utenti che la consultano. Tuttavia, lo spazio web che la ospita potrebbe richiedere una forma di iscrizione o di autenticazione. Se tale iscrizione o autenticazione fosse richiesta, è necessario valutare l'informativa sul trattamento dei dati personali dello spazio web in questione, per sapere come e a che scopo tali dati vengono trattati.

Quest'opera, così come realizzata dal suo autore, non contiene inserzioni pubblicitarie. Tuttavia, lo spazio web che la ospita potrebbe iniettare il codice necessario a somministrare della pubblicità durante la sua consultazione o prima dello scarico dei file. Tali inserzioni pubblicitarie, se ci sono, non hanno nessuna relazione con l'autore di quest'opera e nemmeno vi portano alcun beneficio economico, in quanto servono esclusivamente al mantenimento dello spazio web ospitante.
Le inserzioni pubblicitarie, se presenti, possono utilizzare una tecnologia atta a riconoscere gli accessi che provengono dallo stesso computer o dallo stesso terminale, assieme a tutte le informazioni che possono essere estrapolate dall'origine dell'accesso e sulle funzionalità del computer o del terminale usato per accedere (incluso il fatto che sia disponibile o meno del software che possa essere utile a recepire la pubblicità stessa). Per conoscere il modo in cui le informazioni vengono raccolte dalla pubblicità (se c'è) e il loro utilizzo effettivo, è necessario valutare l'informativa sul trattamento dei dati personali dello spazio web che ospita l'opera.

2

Una copia della licenza GNU General Public License, versione 3, si trova nell'appendice A; una copia della licenza GNU Free Documentation License, versione 1.3, si trova nell'appendice B; una copia della licenza Creative Commons Attribution-ShareAlike, versione italiana 2.5, si trova nell'appendice C.

A copy of GNU General Public License, version 3, is available in appendix A; a copy of GNU Free Documentation License, version 1.3, is available in appendix B; a copy of Creative Commons Attribution-ShareAlike License, italian version 2.5, is available in appendix C.

Per tutti i riferimenti dell'opera si veda *http://informaticalibera.net* . Al momento della pubblicazione di questa edizione, i punti di distribuzione in-linea più importanti, sono presso Internet Archive (*http://www.archive.org/details/AppuntiDiInformaticaLibera*), il GARR (*http://appuntilinux.mirror.garr.it/mirrors/appuntilinux/*), ILS (*http://appunti.linux.it*) e il Pluto (*http://a2.pluto.it*).

3

Parte xiv Applicativi 9
Spreadsheet Calculator 11
Parte xv NLNX 27
Semplicità e controllo 33
Ambiente operativo ideale di NLNX 37
NLNX per il principiante 39
La configurazione grafica di NLNX 43
Accesso ad altri file system con NLNX 57
Sintesi dei comandi 61
Sintesi delle opzioni di avvio 65
Preparazione delle partizioni 69
Copia del sistema operativo 75
Installazione in un file-immagine 79
Predisposizione del sistema di avvio 81
Avvio di NLNX dopo la sua installazione 85
Realizzazione di una propria variante di NLNX 87
Configurazione di NLNX per l'interazione con il sistema 93
Stampa 99
Particolarità varie di NLNX 103
Servizi di rete vari, secondo l'organizzazione di NLNX 109
PXE per l'avvio di un elaboratore senza disco fisso 113
HTTP e servizi collegati 117
La rete e gli instradamenti con NLNX 121
VNC con «nlnxrc» 133
Utenze e amministrazione con NLNX 139
Utenze condivise e configurazione automatica con NLNX 151
Controllo dell'accesso a servizi HTTP esterni 161
Controllo dell'accesso alla stampante 163
Mettere in comunicazione insegnanti e studenti 165
Estendere il controllo delle utenze alla rete MS-Windows 167
Installazione indolore di NLNX in una rete di elaboratori MS-Windows 179
Problemi di NLNX e soluzioni 183
Adattamento di NLNX 187
Organizzazione del laboratorio GNU/Linux 191
Utilizzo del laboratorio GNU/Linux 199
Organizzazione dei laboratori MS-Windows 205
Utilizzo dei laboratori MS-Windows 219
Servizio telnet.informaticalibera.net 225
Come funziona il servizio telnet.informaticalibera.net 231
Parte xvi Installazione e avvio di un sistema GNU/Linux 237
Installazione di una distribuzione Slackware 239
LILO: introduzione 247
Configurazione di LILO più in dettaglio (omesso) 253
Parte xvii X 261
X: monitor, adattatore grafico e frequenza dot-clock 263
Parte xviii Posta elettronica 277
Sendmail: introduzione 279
Exim 3: introduzione 287
Ssmtp 303
Parte xix Usenet 305

Introduzione a Usenet 307
Introduzione a INN -- InterNet News 313
Parte xx Cfengine 329
Introduzione a Cfengine 331
Cfengine: sezioni di uso comune 341
Cfengine attraverso la rete 351
Parte xxi Connettività con sistemi Dos 355
Dos IPv4 357
Dos PPP 371
Introduzione a NOS-KA9Q -- IPv4 per Dos 375
nanoDos 387
Parte xxii ALML 391
Introduzione ad ALML 395
Preparazione e visione generale 397
Il documento secondo Alml 411
Elementi interni alle righe 429
Blocchi comuni 433
Altri blocchi e componenti lineari particolari 443
Riferimenti, note e altre informazioni 449
Immagini e video 459
Tabelle 467
Allegati 471
Verifiche 473
Esempio di verifica con Alml 481
Esempio di verifica con Alml bis 483
Esempio di verifica con Alml ter 485
Presentazioni 487
Inserimento letterale di codice TeX e HTML, con eventuale inserimento condizionato 491
Entità ISO ed entità HTML gestite da Alml 493
Insieme di caratteri universale e Alml 513
Stile di scrittura del sorgente 537
Alml per i grandi progetti di documentazione 541
Questioni tecniche 545
Gestione di «a2» 555
Convenzioni di «a2» 559
Glossario stilistico di «a2» 571
Parte xxiii Texinfo: lo standard della documentazione GNU 601
Introduzione a Texinfo 603
Texinfo: libro e ipertesto 617
Sgmltexi: installazione e utilizzo 625
Sgmltexi: struttura 629
Sgmltexi: contenuti 641
Corrispondenza tra Texinfo e Sgmltexi 653
Parte xxiv Sistemi vari di composizione elettronica 675
Introduzione a Lout 677
Introduzione a LyX 705
Introduzione a HieroTeX 709
Trasformazione in altri formati 721
Parte xxv Circuiti logici 727

Operatori logici e porte logiche 729
Circuiti combinatori 741
Costruzione di un'unità aritmetico-logica 765
Unità aritmetico-logica e registri espandibili 779
Flip-flop 783
Registri 795
Bus e unità di controllo 803
Tkgate 809
Riferimenti 813
Parte xxvi Costruzione di una CPU dimostrativa 815
Versione A: caricamento ed esecuzione del codice 817
Versione B: indice della memoria 831
Versione C: registri generici 837
Versione D: ALU 841
Versione E: indicatori 855
Versione F: condizioni 863
Versione G: pila 869
Versione H: I/O 875
Versione I: ottimizzazione 885
Versione J: ottimizzazione bis 909
Versione K: 16 bit «little-endian» 911
Riferimenti 949
Parte xxvii Pascal 951
Pascal: preparazione di alcuni compilatori comuni 953
Pascal: introduzione 959
Pascal: tipi di dati derivati 973
Pascal: esempi di programmazione 979
Parte xxviii Java 993
Java: preparazione 995
Java: introduzione 1001
Java: programmazione a oggetti 1013
Java: esempi di programmazione 1025
Parte xxix Scheme 1037
Scheme: preparazione 1039
Scheme: introduzione 1045
Scheme: struttura del programma e campo di azione 1063
Scheme: liste e vettori 1069
Scheme: I/O 1075
Scheme: esempi di programmazione 1079
Parte xxx BC: linguaggio aritmetico a precisione arbitraria 1091
BC: esempi di programmazione 1093
Parte xxxi Basic 1101
Basic: introduzione 1103
Basic: esempi di programmazione 1111
Parte xxxii File «.DBF» 1115
File «.DBF»: dBase III e derivati 1117
nanoBase 1997 1121
nanoBase 1997 user manual 1127
Clean the Clipper 5.2 1257
Parte xxxiii Programmare a 16 bit 1283

Microprocessori x86-16 1285
Architettura IBM PC 1307
Strumenti di sviluppo e di utilizzo 1313
Parte xxxiv Studio per un sistema a 16 bit 1319
Introduzione a os16 1321
Caricamento ed esecuzione del kernel 1327
Funzioni interne legate all'hardware 1333
Gestione della memoria 1341
Gestione dei terminali virtuali 1345
Dispositivi 1347
Gestione del file system 1351
Gestione dei processi 1365
Caricamento ed esecuzione delle applicazioni 1377
Parte xxxv Manuale di os16 1381
Avvio del sistema e conclusione 1387
Sezione 1: programmi eseguibili o comandi interni di shell 1391
Sezione 2: chiamate di sistema 1401
Sezione 3: funzioni di libreria 1431
Sezione 4: file speciali 1493
Sezione 5: formato dei file e convenzioni 1499
Sezione 7: varie 1501
Sezione 8: comandi per l'amministrazione del sistema 1503
Sezione 9: kernel 1507
Parte xxxvi Codice di os16 1587
Script e sorgenti del kernel 1589
Sorgenti della libreria generale 1727
Sorgenti delle applicazioni 1849
Parte xxxvii Un sistema operativo giocattolo, denominato «05» 1901
Preparazione 1903
Directory di lavoro 1905
Libreria standard per iniziare 1909
File isolati per dichiarazioni riprese in più librerie 1911
Librerie specifiche generali 1931
Un primo kernel di prova 1939
Tabella GDT 1943
Gestione della memoria 1947
Tabella IDT 1955
Chiamate di sistema 1967
Interruzioni hardware 1971
Una specie di «shell» 1977
Parte xxxviii *BSD 1981
Caratteristiche comuni nei sistemi *BSD 1983
Parte xxxix i86 1985
Minix 1987
ELKS: introduzione 2003
ELKS: realizzazione personale 2007
Parte xl Dos 2017
Dos: introduzione 2019
Dos: dischi, file system, directory e file 2031
Dos: configurazione 2039

Dos: script dell'interprete dei comandi 2045
Dos: gestione della memoria centrale 2051
FreeDOS 2053
Progetto GNUish 2057
The valuable DOS Freeware page 2059
Introduzione a ReactOS 2065
DOSEMU: l'emulatore di hardware DOS compatibile 2067
Parte xli Sistemi operativi alternativi 2075
Sistemi operativi alternativi di un certo rilievo 2077
Syllable: introduzione 2079
Syllable: utilizzo sommario 2085
Plan 9: installazione 2089
UNIX di ricerca 2099

Spreadsheet Calculator 11
Concetti generali 11
Avvio e opzioni di funzionamento 13
Inserimento e modifica dei valori nelle celle 14
Formato delle celle 17
Comandi per intervenire sui file 19
Comandi per intervenire su zone del foglio 20
Comandi vari 21
Espressioni 22
Formato del file 24

Spreadsheet Calculator

Concetti generali 11
Zona 12
Navigazione nel foglio 12
Inserimento e modifica di dati nella riga di comando 13
Avvio e opzioni di funzionamento 13
Inserimento e modifica dei valori nelle celle 14
Particolarità dell'inserimento di valori ed espressioni stringa 16
Particolarità dell'inserimento di valori ed espressioni numerici 16
Formato delle celle 17
Comandi per intervenire sui file 19
Comandi per intervenire su zone del foglio 20
Aiuto alla definizione delle celle e delle zone 21
Coordinate relative o assolute 21
Comandi vari 21
Espressioni 22
Formato del file 24

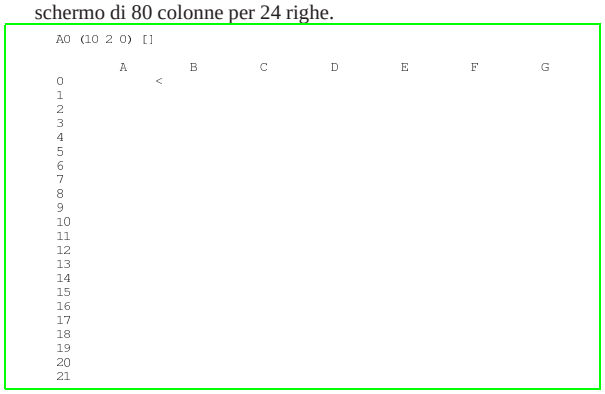
SC, [1] ovvero Spreadsheet Calculator, è un applicativo per l'elaborazione di fogli elettronici di vecchia concezione. La sua origine è molto lontana, a partire da un lavoro apparso su Usenet, modificato successivamente da più autori. Oggi esistono diverse varianti di questo applicativo, ognuna con un nome particolare; è disponibile anche una versione per Dos.

Nonostante si tratti di un foglio elettronico superato sotto vari punti di vista, rimane un applicativo fondamentale nell'informatica del software libero, se non altro per motivi storici.

Concetti generali

In condizioni normali, una volta avviato SC, si vede una schermata simile a quanto riportato in figura u10.1.

Figura u10.1. Aspetto iniziale del foglio elettronico su uno schermo di 80 colonne per 24 righe.



Sullo schermo si distinguono quattro parti. La prima riga in alto serve per visualizzare il contenuto delle celle e per l'inserimento dei comandi da parte dell'utilizzatore. La seconda riga viene usata dal programma per mostrare dei messaggi, che si solito servono per conoscere l'esito di un comando appena impartito. La terza riga e le prime quattro colonne mostrano le coordinate delle celle, dove le righe sono identificate attraverso un numero e le colonne attraverso una lettera. Infine, la parte restante dello schermo è riservata alle celle del foglio elettronico.

Le coordinate delle celle sono indicate usando prima la lettera e poi il numero, per esempio A0. È possibile usare lettere maiuscole o minuscole indifferentemente.

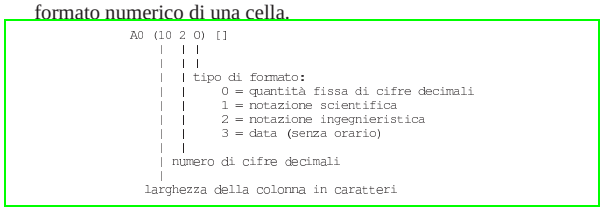
Sono disponibili due cursori: uno per navigare tra le celle; l'altro per scrivere i comandi. Osservando la figura u10.1 si può notare che il cursore del foglio si trova sulla cella A0, alla cui destra appare il simbolo '**<**'. In condizioni normali, la cella è anche evidenziata con un colore adatto.

È interessante interpretare ciò che appare nella prima parte della prima riga dello schermo. Nel caso della figura, si vede:

```
A0 (10 2 0) []
```

Significa che il cursore si trova sulla cella A0, che il formato della cella è espresso in qualche modo dalle cifre «10 2 0» e che si il suo contenuto è nullo (è vuota). Il formato si interpreta nel modo schematizzato nella figura u10.3.

Figura u10.3. Interpretazione delle cifre che descrivono il formato numerico di una cella.



Il formato numerico espresso in questo modo, può essere definito solo su una colonna intera e non si può circoscrivere a un gruppo più limitato di celle. In un gruppo di celle è possibile intervenire successivamente per modificare la rappresentazione dei valori numerici, senza alterare la larghezza della colonna.

Prima di descrivere i comandi di SC è bene sapere subito che un comando non completato può essere annullato premendo il tasto [*Esc*], anche più volte se necessario.

Zona

In tutti gli applicativi per l'elaborazione di fogli elettronici, la zona è un rettangolo di celle che va da un minimo di una sola cella fino al massimo della dimensione del foglio. Per delineare questo rettangolo si indicano le coordinate di due celle che si trovano negli angoli opposti di questo rettangolo. Con SC si usano i due punti (verticali) per unire queste due coordinate.

Per esempio, A0:A0 è la zona che si riduce alla sola cella A0; A0:B1 è una zona di quattro celle totali, composta in pratica da A0, B0, A1 e B1. Di solito, salvo situazioni particolari, l'inversione nell'ordine delle coordinate e l'utilizzo degli altri due estremi non cambia il significato, per cui A0:B1 è uguale a B1:A0, così come è uguale a A1:B0 e a B0:A1.

In certe situazioni, quando si vuole sottolineare il fatto che si sta facendo riferimento a un intervallo di colonne, si usa una forma simile, in cui non appaiono le coordinate numeriche delle righe. per esempio, A:C rappresenta le colonne A, B e C. Nello stesso modo è possibile indicare un intervallo di righe; per esempio, 1:3 indica le righe 1, 2 e 3.

Navigazione nel foglio

La navigazione all'interno del foglio potrebbe non essere così intuitiva come ci si aspetterebbe. Teoricamente, si possono usare i tasti freccia; in pratica, questo potrebbe non essere vero. Quando ci sono difficoltà bisogna affidarsi alla tradizione, ovvero al metodo di VI, con l'aggiunta di qualche altra possibilità. La tabella u10.4 riepiloga i comandi per lo spostamento del cursore nel foglio.

Bisogna considerare che per ottenere l'inserimento di un'informazione in una cella, occorre usare prima il comando apposito che permette di entrare nella fase di inserimento. Pertanto, la pressione «casuale» di un tasto si traduce quasi sempre in un comando.

Tabella u10.4. Comandi principali per la navigazione tra le celle del foglio.

Comando	Alternativa	Spostamento
[*h*]	[*Ctrl b*]	A sinistra di una cella.
[*j*]	[*Ctrl n*]	In basso di una cella.
[*k*]	[*Ctrl p*]	In alto di una cella.
[*l*]	[*Ctrl f*]	A destra di una cella.

Esistono anche una serie di comandi utili per raggiungere posizioni particolari in modo rapido. In particolare, può essere utile raggiungere solo celle che contengono già qualcosa (nella documentazione originale si parla di «celle valide»). La tabella u10.5 riepiloga altri comandi per lo spostamento nel foglio.

Tabella u10.5. Comandi speciali per la navigazione nel foglio.

Comando	Spostamento
[^]	Sulla prima riga della colonna attuale.
[#]	Sull'ultima riga che contiene qualcosa, della colonna attuale.
[*0*]	Sulla prima colonna della riga attuale.
[$]	Sull'ultima colonna che contiene qualcosa, della riga attuale.
[*b*]	Sulla cella precedente che contiene qualcosa.
[*w*]	Sulla cella successiva che contiene qualcosa.
[*g*]***coordinata***	Sulla cella indicata dalla coordinata.

Inserimento e modifica di dati nella riga di comando

L'aspetto più complicato per un principiante alle prese con questo applicativo è probabilmente la modifica del testo nella riga di comando. Ci sono varie situazioni in cui occorre inserire un'informazione; di solito si tratta di introdurre il valore di una cella, oppure si deve completare un comando specificando una zona, o altro. In queste situazioni, si accede alla parte superiore dello schermo, in quella che qui viene chiamata la riga di comando.

Quando il contesto porta a inserire qualcosa, tutto avviene come ci si potrebbe aspettare, inserendo normalmente il testo, dove il tasto [*backspace*] funziona regolarmente per cancellare. Tuttavia, la cosa non è così semplice come appare, perché si tratta di una riga che riconosce i comandi di VI: è sufficiente premere [*Esc*] per passare alla modalità di comando. La tabella u10.6 riepiloga le funzionalità più importanti della modalità di comando, quando si sta lavorando sulla prima riga dello schermo.

Tabella u10.6. Comandi disponibili quando si modifica qualcosa sulla prima riga dello schermo.

Comando	Descrizione
[*i*]	Termina la modalità di comando e inizia l'inserimento.
[*a*]	Termina la modalità di comando e inizia l'inserimento dopo il cursore.
[*h*]	Va a sinistra di un carattere.
[*l*]	Va a destra di un carattere.
[*x*]	Cancella il carattere corrispondente al cursore.
[*Invio*]	Conferma e conclude.

Avvio e opzioni di funzionamento

L'avvio di questo programma è molto semplice. Sono disponibili alcune opzioni che hanno dei comandi corrispondenti in fase di funzionamento del programma.

```
sc [opzioni] [file]
```

La tabella u10.7 riepiloga le opzioni più importanti. Successivamente vengono mostrate altre tabelle contenenti la descrizione di

comandi interattivi.

Tabella u10.7. Opzioni principali da dare all'avvio dell'eseguibile.

Opzione	Descrizione
`-r`	Ricalcola per righe (predefinito).
`-c`	Ricalcola per colonna.
`-m`	Disabilita il ricalcolo automatico.
`-n`	Inserimento numerico rapido.

Durante il funzionamento sono disponibili due gruppi di comandi per modificare l'impostazione del programma. Si tratta di comandi che scambiano alcune modalità, attivandole o disattivandole, in funzione dello stato precedente, oppure di comandi che impostano in modo preciso. I comandi che scambiano le modalità iniziano con la combinazione [*Ctrl t*] e seguono con un'altra lettera; i comandi di impostazione iniziano con la lettera '**S**' (maiuscola), che richiede poi l'inserimento di un comando scritto per esteso. Per esempio, la sequenza [*Ctrl t*][*a*] attiva o disattiva la modalità di ricalcolo automatico; nello stesso modo, il comando [*S*] seguito da '**iterations=7**' stabilisce che le iterazioni da eseguire per ricalcolare i valori delle espressioni devono essere sette. La tabella u10.8 riepiloga i comandi più importanti da impartire in questo modo.

Tabella u10.8. Alcuni comandi che modificano la modalità di funzionamento, distinguendo tra quelli di scambio, che iniziano con [*Ctrl t*] e quelli di impostazione, che iniziano con [*S*].

Comando	Descrizione
[*Ctrl t*][*a*]	Attiva o disattiva il ricalcolo automatico delle celle.
[*Ctrl t*][*c*]	Attiva o disattiva l'evidenziamento della cella corrente.
[*Ctrl t*][*n*]	Attiva o disattiva la modalità di inserimento numerico rapido.
[*Ctrl t*][*t*]	Attiva o disattiva la visualizzazione di informazioni sulla prima riga.
`S byrows`	Definisce un ricalcolo delle celle per righe.
`S bycols`	Definisce un ricalcolo delle celle per colonne.
`S iterations=`***n***	Definisce il numero massimo di cicli per il ricalcolo delle celle.
`S tblstyle=0`	Esporta la tabella con i campi separati da due punti.
`S tblstyle=tbl`	Esporta la tabella in formato Tbl (*roff).
`S tblstyle=latex`	Esporta la tabella in formato LaTeX.
`S tblstyle=tex`	Esporta la tabella in formato TeX.
`S tblstyle=frame`	Esporta la tabella in formato FrameMaker.

È opportuno sottolineare che le impostazioni di esportazioni si riferiscono al comando '**T**', con il quale si esporta il contenuto della tabella in un altro formato.

Inserimento e modifica dei valori nelle celle

Questo tipo di elaboratore di fogli elettronici, distingue tra diversi contesti di funzionamento. Inizialmente ci si trova in una modalità di comando, dove molti dei tasti (lettere o numeri) possono rappresentare l'inizio di un comando. In base a questa logica, l'inserimento dei dati nelle celle deve essere preceduto da un comando apposito: '**"**' inizia l'inserimento di una cella che rappresenta un valore stringa; '**=**' inizia l'inserimento di una cella che rappresenta un valore numerico.

Tale premessa permette di comprendere che questo tipo di foglio elettronico è in grado di gestire soltanto due tipi di dati: stringhe e numeri (reali). Questi valori possono essere inseriti in forma costante, oppure si possono indicare delle espressioni che generano un risultato del tipo previsto.

L'uso del comando '**=**' o '**"**' porta il programma in un contesto di funzionamento particolare, in cui si digita l'espressione (costante o meno) sulla prima riga dello schermo, che mostra l'invito '**i>**' (*insert*).

```
i>
```

Tale digitazione è sottoposta al controllo che è già stato descritto in precedenza: è come se fosse stato iniziato un comando di inserimento con il programma VI, dal quale, se si preme il tasto [*Esc*] si passa alla modalità di comando relativa, che viene fatta notare attraverso l'invito '**e>**' (*edit*).

```
e>
```

Questo è riassunto nella tabella u10.6 che è già stata mostrata in precedenza. L'informazione viene memorizzata nella cella solo se si conclude l'inserimento con la pressione di [*Invio*]; pertanto, se si passa alla modalità di comando (attraverso il tasto [*Esc*]) e poi si preme nuovamente [*Esc*], si annulla l'inserimento.

L'inserimento di qualcosa in una cella, viene filtrato attraverso un comando di un linguaggio interno al foglio elettronico. Se si assegna una stringa, il comando è:

`label` ***cella*** `=` {***stringa_delimitata*** | ***espressione_stringa***}

Se invece si assegna un valore numerico, il comando è:

`let` ***cella*** `=` {***costante_numerica*** | ***espressione_numerica***}

Quando si inserisce un valore, bisogna fare attenzione a non modificare la parte iniziale del comando, a meno che si tratti di un'azione voluta.

Per modificare il contenuto di una cella, in generale è possibile cancellarlo prima, per poi inserire quello nuovo, dove la cancellazione si ottiene con il comando '**x**'. La modifica del contenuto richiede invece di passare a un contesto di funzionamento analogo a quello di inserimento, in cui però ci si trova inizialmente in modalità di modifica. In pratica, ci si trova subito di fronte all'invito '**e>**' e ci si deve comportare di conseguenza.

Per passare alla modifica del contenuto di una cella, si usano due comandi differenti, a seconda che si intenda modificare un valore stringa, '**E**', o un valore numerico, '**e**'. Questo serve a cambiare il comando interno del foglio elettronico, che nel primo caso è '**label**', mentre nel secondo è '**let**'. Tutto questo ha un significato che si chiarisce in seguito, quando viene descritto il formato in cui vengono salvati i dati.

Per il momento, è importante comprendere che una cella può contenere simultaneamente due valori: uno numerico e uno stringa. In generale, una cosa del genere non dovrebbe essere ammissibile, ma la logica di SC porta a tale situazione.

Tabella u10.9. Comandi di inserimento e modifica dei valori delle celle. I comandi particolari che sono disponibili durante il contesto di modifica, sono già stati descritti in un'altra tabella.

Comando	Descrizione
[=]	Inizia l'inserimento di un valore numerico.
["]	Inizia l'inserimento di un valore stringa (centrato).
[<]	Inizia l'inserimento di un valore stringa a sinistra.
[>]	Inizia l'inserimento di un valore stringa a destra.
[*e*]	Inizia la modifica di un valore numerico.
[*E*]	Inizia la modifica di un valore stringa.

Particolarità dell'inserimento di valori ed espressioni stringa

L'inserimento di una stringa, inizia con il comando '**"**', che porta ad attivare la riga di comando sulla prima riga dello schermo. Supponendo di intervenire sulla cella A7, si ottiene questo:

```
i> label A7 = "
```

Il cursore per la digitazione della stringa si trova subito dopo gli apici doppi che si vedono nell'esempio. Quello che si sta inserendo è un'istruzione del foglio elettronico, con la quale si assegna la stringa (*label*) alla cella A7. Si intuisce che gli apici doppi iniziali servono a delimitare una stringa costante. Supponendo di voler inserire la parola «ciao», si può procedere come di seguito:

```
i> label A7 = "ciao
```

Alla fine, si può concludere la stringa con un altro apice doppio, oppure si può anche farne a meno. In ogni caso, al termine si conclude con un [*Invio*].

La stringa può essere collocata al centro della cella, a sinistra o a destra. Il comando '**"**' inizia l'inserimento di una stringa centrata; per l'allineamento a sinistra si usa il comando '**<**'; per l'allineamento a destra si usa il comando '**>**'. A questi comandi corrispondono altrettante istruzioni interne del foglio elettronico, che vengono generate automaticamente:

Comando	Descrizione
`label` ***cella*** `=` ***espressione***	stringa centrata;
`leftstring` ***cella*** `=` ***espressione***	stringa allineata a sinistra;
`rightstring` ***cella*** `=` ***espressione***	stringa allineata a destra.

Fino a questo punto è stato mostrato l'inserimento di stringhe costanti. Per quanto riguarda le espressioni che generano un risultato stringa, bisogna considerare che l'apice doppio iniziale, inserito automaticamente, deve essere eliminato. Per esempio, se la cella A0 contiene una stringa, volendo fare riferimento al suo contenuto nella cella B3, si deve usare il riferimento alla cella A0, senza delimitazioni. In pratica, all'inizio si ha questa situazione:

```
i> label B3 = "
```

Quindi, si cancella l'apice doppio e si inserisce l'espressione desiderata:

```
i> label B3 = A0 [Invio]
```

Probabilmente, in una stringa costante è impossibile indicare un apice doppio; inoltre, si può usare una barra obliqua iniziale per ottenere la ripetizione della parte successiva, per tutta la larghezza della colonna. Si osservi l'esempio:

```
i> label A10 = "\-= [Invio]
```

Questo si traduce in pratica nel mostrare la stringa '**-=-=-=-=-=**' all'interno della cella.

Particolarità dell'inserimento di valori ed espressioni numerici

L'inserimento di valori numerici non presenta situazioni particolari. Infatti, un valore numerico costante non richiede alcuna delimitazione, così come le espressioni che generano un risultato numerico. A questo proposito, si può valutare la possibilità di abilitare l'inserimento numerico rapido, con il comando [*Ctrl t*][*n*], oppure attraverso l'opzione '**-n**'.

In generale, per coerenza, non è il caso di intervenire in questo modo; tuttavia, di fronte alla necessità di inserire un gran numero di costanti numeriche, può essere conveniente questo approccio.

Formato delle celle

La gestione del formato delle celle è piuttosto strana, per cui è necessario trattare l'argomento in modo particolare.

Per prima cosa occorre considerare la larghezza della colonna che viene determinata attraverso il comando '**f**'. Questo attende l'inserimento di tre valori numerici: la larghezza in caratteri, la quantità di cifre decimali e il tipo. Questa cosa è già stata anticipata nella figura u10.3, in cui è descritto dettagliatamente il significato dell'ultimo valore.

Per fare un esempio molto semplice, il formato '**10 2 0**', che è quello predefinito, rappresenta una colonna di 10 caratteri di larghezza, in cui i valori vengono rappresentati con due decimali, attraverso una notazione normale a virgola fissa. In pratica, in queste condizioni, si possono rappresentare numeri da '**-999999.99**' a '**9999999.99**'.

Anche la notazione scientifica e quella ingegnieristica, definite rispettivamente dal tipo uno e due, utilizzano l'informazione sulla quantità di decimali. Al contrario, il formato delle date (il numero tre), non dipende dalla quantità dei decimali.

In pratica, il comando '**f**' definisce il formato numerico generale di una colonna, cosa che si traduce anche nella definizione della larghezza della colonna stessa. Evidentemente, per quanto riguarda le stringhe, queste risentono solo della larghezza della cella e non delle altre informazioni.

Nell'ambito di una sola cella, è possibile cambiare il formato generale della colonna attraverso il comando '**F**', mentre per un gruppo di celle si usa il comando analogo '**/F**'. Questo richiede l'inserimento di una stringa speciale, composta da caratteri che servono a rappresentare un formato numerico. Il significato di questi simboli appare descritto negli schemi seguenti.

Tabella u10.11. Comandi per il controllo del formato delle colonne e della visualizzazione dei valori numerici.

Comando	Descrizione
`f` ***larghezza decimali tipo***	Formato generale della colonna.
`F` ***stringa_formato***	Formato particolare della cella.
`/F` ***zona stringa_formato***	Formato particolare di una zona di celle.

Tabella u10.12. Stringa di definizione del tipo per il primo modello.

Simbolo	Descrizione
`0`	Virgola fissa.
`1`	Notazione scientifica.
`2`	Notazione ingegnieristica.
`3`	Data.

Tabella u10.13. Stringa di definizione del formato, per il secondo e il terzo modello.

Simbolo	Descrizione
`#`	Cifra numerica eventuale.
`0`	Cifra numerica fissa.
`.`	Separatore decimale.
`,`	Separatore delle migliaia.
`%`	Inserisce il simbolo e mostra in forma di percentuale.
`\`***x***	Tratta ***x*** in modo letterale.
`E+` | `e+`	Notazione scientifica.

Simbolo	Descrizione
`E-` | `e-`	Notazione scientifica mostrando il segno solo quando negativo.
`;`	Divide due formati per i valori positivi e i valori negativi.

Si possono osservare i comandi interni del foglio elettronico nel momento in cui si definisce il formato della colonna, oppure il formato specifico di un gruppo di celle. Nel primo caso si tratta dell'istruzione '**`format`**':

`format` ***colonna larghezza decimali tipo***

Nel secondo caso, l'istruzione si abbrevia:

`fmt` ***zona stringa_di_formato_delimitata***

Nel seguito vengono mostrati alcuni esempi, cercando di riprodurre nel modo migliore possibile la situazione che si vede sullo schermo. Si deve tenere presente che l'istruzione interna del foglio elettronico viene generata automaticamente; quello che serve, semmai, è di fare attenzione a non cancellarla.

L'informazione che appare tra parentesi quadre nell'istruzione del foglio elettronico ('**`[for column]`**' e '**`[format]`**'), non fa parte dell'istruzione stessa. Viene collocata per facilitare all'utilizzatore la comprensione dell'azione che si sta compiendo.

I comandi '**`F`**' e '**`/F`**' sono identici dal punto di vista del foglio elettronico, perché generano la stessa istruzione interna. Nel primo caso, la zona viene indicata automaticamente, riferendola alla cella corrente.

Segue la descrizione di alcuni esempi.

- Modifica il formato della colonna A, in modo da avere 15 caratteri di larghezza, riservando tre cifre per i decimali, mostrando valori a virgola fissa:

  [*f*]

  `i>` **`format [for column] A 15 3 0`** [*Invio*]

- Come nell'esempio precedente, utilizzando però una notazione scientifica:

  [*f*]

  `i>` **`format [for column] A 15 3 1`** [*Invio*]

- Modifica il formato particolare della cella A1, in modo da rappresentare valori che vanno da un minimo di -9999999,999 a un massimo di 99999999,999:

  [*F*]

  `i>` **`fmt [format] A1 "########.000`** [*Invio*]

- Modifica il formato particolare della cella A1, in modo da rappresentare valori percentuali interi.

  [*F*]

  `i>` **`fmt [format] A2 "###%`** [*Invio*]

- Modifica il formato particolare della cella A3, in modo da rappresentare valori che vanno da un minimo di -99999999,999 a un massimo di 99999999,999; in particolare, i valori negativi sono rappresentati tra parentesi:

  [*F*]

  `i>` **`fmt [format] A3 "########.000;\(########.000\)`** [*Invio*]

Comandi per intervenire sui file

Per caricare o salvare il foglio, si interviene con comandi composti da una lettera, seguita dal nome del file e forse da altre indicazioni. Vengono descritti brevemente questi comandi.

- `G` ***file_da_caricare***

  Il comando '**`G`**' sta per *Get* e permette di caricare un foglio elettronico che in precedenza è stato salvato su un file. Appena si preme la lettera '**`G`**', si passa sulla parte superiore dello schermo a scrivere il nome di tale file, che deve essere conosciuto preventivamente, perché non viene dato alcun ausilio di ricerca:

  `i>` **`get ["source"] "prova`** [*Invio*]

  L'esempio mostra in pratica il completamento del comando per caricare il file '`prova`' che si trova nella directory corrente.

- `M` ***file_da_caricare***

  Il comando '**`M`**' sta per *Merge* e permette di caricare un altro foglio elettronico che vada a sommarsi a quanto appare già sullo schermo. In pratica, le celle che dovessero già contenere qualcosa, vengono sovrascritte, perdendo l'informazione precedente.

- `P` [[***file_da_salvare***] ***zona***]

  Il comando '**`P`**' sta per *Put* e permette di salvare il foglio elettronico in un file. Appena si preme la lettera '**`P`**', si passa sulla parte superiore dello schermo a scrivere il nome di tale file, che però non è necessario se in precedenza il foglio è già stato salvato.

  `i>` **`put ["dest" range] "prova`** [*Invio*]

  L'esempio mostra in pratica il completamento del comando per salvare il foglio nel file '`prova`'. Se il nome è già stato stabilito in precedenza, è sufficiente premere [*Invio*] per confermare l'operazione con il nome precedente.

  Volendo, è possibile salvare solo una parte del foglio elettronico, definendo le coordinate della zona a cui si è interessati.

  `i>` **`put ["dest" range] "prova-1" A0:F10`** [*Invio*]

  In questo caso, si vuole salvare la zona delimitata dalle coordinate A0:F10 nel file '`prova-1`'. Si noti che in questo caso è stato necessario concludere la delimitazione del nome del file attraverso gli apici doppi finali.

- `W` [[***file_di_testo***] ***zona***]

  Il comando '**`W`**' sta per *Write* e si comporta in modo analogo a '**`P`**', con la differenza che genera un file di testo contenente la rappresentazione finale del foglio. In pratica, si tratta di una forma di esportazione che potrebbe essere diretta anche alla stampa.

  `i>` **`write ["dest" range] "| lpr" A0:F10`** [*Invio*]

  Con questo comando si vuole stampare la zona A0:F10, attraverso l'invio della stessa al comando '**`lpr`**'.

- `T` [[***file_da_esportare***] ***zona***]

  Il comando '**`T`**' sta per *Table* e serve a esportare una zona, o tutto il foglio, in un formato differente, definito attraverso il comando '**`S tblstyle`**', già descritto nella tabella u10.8. Il funzionamento è analogo ai comandi '**`P`**' e '**`W`**'.

Tabella u10.14. Comandi per la gestione dei file.

Comando	Descrizione
`G` ***file***	Carica il file.
`M` ***file***	Sovrappone il file al foglio attuale.

Comando	Descrizione
`P` [*file* [*zona*]]	Salva il foglio o solo una zona particolare.
`W` [*file* [*zona*]]	Salva (esporta) in formato testo.
`T` [*file* [*zona*]]	Esporta in un altro formato.

In generale, tutti i comandi che servono a salvare o a esportare dati, permettono l'indicazione di un file su disco, oppure di un comando del sistema operativo da alimentare attraverso un condotto. In pratica, se il nome del file inizia con il simbolo '**|**', si intende che si tratti di un condotto.

Comandi per intervenire su zone del foglio

Per qualche ragione, i comandi che intervengono su una zona rettangolare del foglio, iniziano tutti con la barra obliqua normale ('**/**') e continuano con una lettera, dopo la quale, di solito, si è invitati a inserire la zona a cui si fa riferimento.

Da questo si intende che le coordinate delle zone vanno scritte sempre dopo aver iniziato il comando, nel modo che è già stato mostrato:

coordinata_iniziale* : *coordinata_finale

In alternativa, si può indicare il nome della zona, che eventualmente gli fosse stato assegnato in precedenza con il comando '**/d**'.

Il programma offre anche qualche accorgimento per facilitare l'inserimento delle zone, ma in generale non si tratta di soluzioni convenienti, per cui di solito è meglio usare la digitazione normale. La tabella u10.15 riepiloga brevemente i comandi principali di questo tipo, mentre nel seguito sono mostrati alcuni esempi, in cui si vede anche l'istruzione interna del foglio elettronico, che comunque viene generata automaticamente.

Tabella u10.15. Comandi per la gestione delle zone del foglio.

Comando	Descrizione
`/x` ***zona***	Cancella il contenuto delle celle nella zona indicata.
`/c` ***zona_destinazione zona_origine***	Copia una zona.
`/c` ***zona_destinazione cella_origine***	Copia una cella riempiendo una zona.
`/f` ***zona_destinazione n_iniziale increm***	Riempie una zona di valori.
`/d` ***stringa_nome zona***	Assegna un nome a una zona di celle.
`/u` ***zona***	Cancella il nome assegnato alla zona in precedenza.
`/s`	Mostra l'elenco delle zone che hanno un nome.
`/l` ***zona***	Impedisce la modifica della zona.
`/U` ***zona***	Libera la zona indicata che così può essere modificata.
`/F` ***zona stringa_formato***	Assegna un formato a una zona di celle.

Segue la descrizione di alcuni esempi.

- Elimina il contenuto delle celle che si trovano nella zona delimitata da A0:F20:
  [*/*][*x*]
  `i> erase A0:F20` [*Invio*]
- Copia la zona D0:D7 in A0:A7:
  [*/*][*c*]
  `i> copy A0:A7 D0:D7` [*Invio*]
- Copia la cella D0 in tutta la zona A0:A7:
  [*/*][*c*]
  `i> copy A0:A7 D0` [*Invio*]
- Riempie le celle della zona B0:B7 a partire dal numero 10, per continuare con numeri che si incrementano di due unità ogni volta:
  [*/*][*f*]
  `i> fill B0:B7 10 2` [*Invio*]
- Assegna alla zona B0:B7 il nome '**elenco**':
  [*/*][*d*]
  `i> define "elenco" B0:B7` [*Invio*]
- Mostra l'elenco delle zone che hanno un nome, inviando l'elenco sotto il controllo di Less o di un altro programma analogo secondo quanto indicato nella variabile di ambiente '**PAGER**'. Per uscire dalla visualizzazione, occorre usare i comandi di quel programma. Con Less basta premere la lettera '**q**'.
  [*/*][*s*]
- Assegna alla zona B0:B7 il formato '**####.0000**':
  [*/*][*F*]
  `i> fmt B0:B7 "####.0000"` [*Invio*]

Aiuto alla definizione delle celle e delle zone

Una volta entrati nell'idea di funzionamento di questo tipo di foglio elettronico, le cose non sono più tanto difficili e può essere utile sfruttare qualche accorgimento che facilita l'inserimento di coordinate riferite a celle o zone. Nel momento in cui ci si trova nella fase di inserimento, nella prima riga dello schermo, sono disponibili alcuni comandi. In particolare i comandi [*Ctrl b*], [*Ctrl n*], [*Ctrl p*] e [*Ctrl f*] sono ancora disponibili, per evidenziare una zona nel foglio sottostante.

Usando questi comandi, la zona evidenziata diventa la zona «predefinita», che si può inserire automaticamente con il tasto [*Tab*]. Bisogna provare un po' e poi si comprende il senso di questo.

Tabella u10.16. Comandi speciali disponibili durante la digitazione nella prima riga dello schermo.

Comando	Effetto
[*Ctrl b*]	*Back*, estende a sinistra la zona predefinita.
[*Ctrl n*]	*Next*, estende in basso la zona predefinita.
[*Ctrl p*]	*Previous*, estende in alto la zona predefinita.
[*Ctrl f*]	*Forward*, estende a destra la zona predefinita.
[*Tab*]	Inserisce la zona predefinita nella riga.
[*Ctrl v*]	Inserisce la cella corrente nella riga.

Coordinate relative o assolute

Le espressioni del foglio elettronico vengono descritte in seguito. Tuttavia, in questa fase è importante rendersi conto della rappresentazione delle celle e delle zone di celle, che in pratica sono le variabili di un foglio elettronico.

Quando un'espressione contiene un riferimento a una cella o a una zona, se questa cella viene copiata in un'altra posizione, questi riferimenti vengono modificati in modo relativo. Per esempio, se la cella A0 contiene un riferimento alla cella B1, copiando la cella A0 in C2, il riferimento interno alla cella C2 punta alla cella D3. Per indicare un riferimento assoluto, si aggiunge nelle coordinate il simbolo '**$**', davanti alla componente che si vuole «bloccare». Si osservino gli esempi seguenti:

- '**X4**' riferimento relativo alla cella X4;
- '**X4**' riferimento assoluto alla cella X4;
- '**$X4**' riferimento alla cella X4, dove la colonna è un'informazione assoluta, mentre la riga rimane un'indicazione relativa;
- '**X$4**' riferimento alla cella X4, dove la riga è un'informazione assoluta, mentre la colonna rimane un'indicazione relativa;

Comandi vari

Nelle sezioni precedenti sono stati esclusi alcuni comandi. In particolare non è ancora stato spiegato come si termina il lavoro con questo programma: ciò si ottiene con [*Q*], [*q*], oppure [*Ctrl c*]; se il foglio attuale non è stato salvato viene chiesto se si vogliono salvare i dati, oppure se si intende rinunciare. La tabella u10.17 riepiloga questi comandi.

Tabella u10.17. Comandi vari.

Comando	Descrizione
[*Q*], [*q*], [*Ctrl c*]	Conclude il funzionamento del programma.
[*Esc*], [*Ctrl g*]	Annulla il comando in corso.
[*Ctrl l*]	Ridisegna lo schermo.
[*Ctrl r*]	Evidenzia le celle che contengono costanti numeriche.
[*Ctrl x*]	Evidenzia le celle che contengono espressioni.
[*@*]	Ricalcola le espressioni.
[*m*]	Copia il contenuto della cella in un'area transitoria.
[*c*]	Incolla il contenuto dell'area transitoria nella cella corrente.
[+]	In condizioni normali, incrementa il valore numerico della cella.
[−]	In condizioni normali, decrementa il valore numerico della cella.
[*Invio*]	Passa all'inserimento di un'istruzione libera.
[*?*]	Mostra la guida interna.

Espressioni

Le celle del foglio sono fatte per contenere delle espressioni. Queste possono essere semplicemente dei valori costanti, numerici o stringa, oppure può trattarsi di qualcosa di più complesso. Le espressioni si ottengono attraverso l'uso di operatori e anche attraverso funzioni che hanno la caratteristica di iniziare con il simbolo '@'. Nel seguito vengono mostrate alcune tabelle che riassumono gli operatori e le funzioni di uso più comune. Resta sempre la documentazione originale per conoscere le altre possibilità a disposizione.

Si può osservare che è possibile rappresentare un risultato booleano. In pratica, *Vero* corrisponde al valore numerico uno; *Falso* corrisponde al valore numerico zero.

Le espressioni possono essere raggruppate attraverso l'uso di parentesi tonde, più o meno annidate.

Tabella u10.18. Elenco degli operatori utilizzabili in presenza di valori numerici e logici.

Espressione	Descrizione
`-`***op***	Inverte il segno dell'operando.
op1*`+`*op2	Somma i due operandi.
op1*`-`*op2	Sottrae dal primo il secondo operando.
op1*`*`*op2	Moltiplica i due operandi.
op1*`/`*op2	Divide il primo operando per il secondo.
op1*`%`*op2	Modulo: il resto della divisione tra il primo e il secondo operando.
op1*`^`*op2	Eleva il primo operando alla potenza del secondo.
op1*`<`*op2	*Vero* se il primo operando è minore del secondo.
op1*`>`*op2	*Vero* se il primo operando è maggiore del secondo.
op1*`<=`*op2	*Vero* se il primo operando è minore o uguale al secondo.
op1*`>=`*op2	*Vero* se il primo operando è maggiore o uguale al secondo.
op1*`=`*op2	*Vero* se gli operandi si equivalgono.
op1*`!=`*op2	*Vero* se gli operandi sono differenti.
`~`***op***	NOT, inversione logica.
op1*`&`*op2	AND logico.
op1*`|`*op2	OR logico.
op1*`?`*op2*`:`*op3	Restituisce il secondo operando se il primo è *Vero*, altrimenti il terzo.

Tabella u10.19. Elenco di alcuni operatori e di alcune funzioni che hanno a che fare con le stringhe.

Espressione	Descrizione
str1*` # `*str2	Concatena le due stringhe.
`@substr(`***str1***`,`***n***`,`***m***`)`	La sottostringa dalla posizione ***n*** per ***m*** caratteri.
`@fmt(`***stringa_printf***`,`***n***`)`	Converte un numero in stringa, in base al formato indicato.
`@upper(`***str1***`)`	Converte in maiuscolo.
`@capital(`***str1***`)`	La prima lettera di ogni parola in maiuscolo.
`@eqs(`***str1***`,`***str2***`)`	*Vero* se le due stringhe sono uguali.

Tabella u10.20. Elenco di alcune funzioni che intervengono su zone.

Espressione	Descrizione
`@sum(`***zona***`)`	Sommatoria dei valori della zona.
`@prod(`***zona***`)`	Prodotto dei valori delle celle.
`@avr(`***zona***`)`	Media aritmetica di tutti i valori validi.
`@count(`***zona***`)`	Quantità di celle contenenti valori validi.
`@max(`***zona***`)`	Valore massimo.
`@min(`***zona***`)`	Valore minimo.

Tabella u10.21. Elenco di alcune funzioni matematiche comuni.

Espressione	Descrizione
`@sqrt(`***n***`)`	Radice quadrata.
`@exp(`***n***`)`	Funzione esponenziale.
`@ln(`***n***`)`	Logaritmo naturale.
`@log(`***n***`)`	Logaritmo a base 10.
`@floor(`***n***`)`	L'intero più grande non superiore del valore fornito.
`@ceil(`***n***`)`	L'intero più piccolo non inferiore del valore fornito.
`@rnd(`***n***`)`	Arrotonda all'intero.
`@round(`***n***`,`***m***`)`	Arrotonda ***n*** all'***m***-esima cifra decimale.
`@abs(`***n***`)`	Valore assoluto.
`@pow(`***n***`,`***m***`)`	***n*** elevato alla potenza di ***m***.
`@pi`	P-greco.
`@dtr(`***n***`)`	Converte da gradi a radianti.
`@rtd(`***n***`)`	Converte da radianti a gradi.
`@sin(`***n***`)`	Seno.
`@cos(`***n***`)`	Coseno.
`@tan(`***n***`)`	Tangente.

Espressione	Descrizione
`@asin(`***n***`)`	Arco-seno.
`@acos(`***n***`)`	Arco-coseno.
`@atan(`***n***`)`	Arco-tangente.

Tabella u10.22. Elenco di alcune funzioni relative a date e orari.

Espressione	Descrizione
`@now`	Tempo attuale in secondi (dal riferimento del 31 dicembre 1969, ore 24:00).
`@dts(`***mese***`,`***giorno***`,`***anno***`)`	Tempo in secondi corrispondente alla data fornita.
`@tts(`***ore***`,`***minuti***`,`***secondi***`)`	Orario in secondi a partire dalla mezzanotte.
`@date(`***tempo_in_secondi***`)`	Restituisce la data corrispondente in forma di stringa.
`@year(`***tempo_in_secondi***`)`	Restituisce l'anno.
`@month(`***tempo_in_secondi***`)`	Restituisce il mese.
`@day(`***tempo_in_secondi***`)`	Restituisce il giorno.
`@hour(`***orario_in_secondi***`)`	Restituisce l'ora.
`@minute(`***orario_in_secondi***`)`	Restituisce i minuti.
`@second(`***orario_in_secondi***`)`	Restituisce i secondi.

Formato del file

«

I file gestiti da SC sono file di testo contenenti direttive del linguaggio interno al foglio elettronico. Per fare un esempio iniziale, basti pensare a un foglio in cui siano state assegnate solo le celle A0 e B1, come si vede nell'esempio:

```
        A       B       C       D       E       F       G
0     123.00
1             ciao   <
```

In pratica, la cella A0 contiene il numero 123, mentre la cella B1 contiene la stringa '**`ciao`**'. Tutto il resto si considera inutilizzato. Salvando questo lavoro in un file, si ottiene ciò che segue:

```
# This data file was generated by the Spreadsheet Calculator.
# You almost certainly shouldn't edit it.

let A0 = 123
label B1 = "ciao"
goto B1
```

Si intuisce che le righe che iniziano con il simbolo '**`#`**', assieme a quelle che sono bianche o semplicemente vuote, vengono ignorate. Tutto il resto viene definito in forma di direttiva, corrispondente a istruzioni del foglio elettronico. In particolare, è stata anche memorizzata la posizione del cursore, che si presume si trovasse sulla cella B1.

Nell'ambito delle direttive, ciò che appare racchiuso tra parentesi quadre viene considerato un commento; di conseguenza viene ignorato. Per esempio, se si modifica il file a mano, nel modo seguente, tutto funziona regolarmente, senza che queste cose influiscano:

```
# This data file was generated by the Spreadsheet Calculator.
# You almost certainly shouldn't edit it.

let [ciao ciao] A0 = 123
label B1 = "ciao" [ciao ciao]
goto B1
```

Negli esempi che sono già stati mostrati nel capitolo, al riguardo di queste istruzioni interne del foglio elettronico, sono già apparse indicazioni tra parentesi quadre, usate per suggerire all'utente le informazioni da inserire. Evidentemente, tali indicazioni non fanno parte delle istruzioni.

La tabella u10.26 riepiloga le istruzioni principali del foglio elettronico, con le quali si può realizzare direttamente un file per SC. Altre istruzioni possono essere individuate semplicemente osservando il comportamento di questo programma.

Tabella u10.26. Alcune istruzioni interne del foglio elettronico, utili per la realizzazione diretta dei suoi file.

Espressione	Descrizione
`let` ***cella*** `=` ***espr_num***	Assegna alla cella un'espressione numerica.
`label` ***cella*** `=` ***espr_str***	Assegna alla cella un'espressione stringa.
`format` ***colonna larghezza decimali tipo***	Modifica il formato di una colonna.
`fmt` ***zona str_formato***	Modifica il formato di una zona.
`set byrows`	Ricalcola per righe.
`set bycols`	Ricalcola per colonne.
`set iterations =` ***n_iterazioni***	Iterazioni per il ricalcolo.
`set tblstyle =` ***stile***	Definisce lo stile di esportazione delle tabelle.
`set autocalc`	Ricalcola automaticamente.
`set !autocalc`	Non ricalcola automaticamente.
`set numeric`	Inserimento numerico rapido.
`set !numeric`	Inserimento numerico normale.
`set cellcur`	Evidenzia la cella corrente.
`set !cellcur`	Non evidenzia la cella corrente.
`set toprow`	Mostra le informazioni sulla prima riga.
`set !toprow`	Non mostra le informazioni sulla prima riga.
`goto` ***cella***	Porta il cursore sulla cella indicata.

[1] **SC** dominio pubblico

Semplicità e controllo 33
Caratteristiche generali 33
Insieme dei pacchetti applicativi disponibili 34
File system compresso 34
Espulsione del disco 34
Aggiornamento 34
Sistema di emergenza 34
Installazione: normale o in sola lettura 34
Avvio dalla rete 35
Ambiente operativo ideale di NLNX 37
Servente 37
Elaboratori periferici 37
Accesso alle stampanti 38
Proxy HTTP 38
Il lavoro a casa 38
Osservazioni 38
NLNX per il principiante 39
Utilizzo del DVD *live* 39
Installazione in «C:\» 39
Utenze 40
Configurazione automatica della rete 41
Voci per l'avvio 41
La configurazione grafica di NLNX 43
Menù 43
Configurazione delle finestre 44
Comandi da tastiera 46
Configurazione locale 47
Configurazione della mappa della tastiera 47
Livelli di inserimento 48
Accenti morti 48
Messaggi di errore 50
Avvio dei programmi 51
Chiusura dei programmi 51
Configurazione del mouse 53
Sfondo 53
Cattura dell'immagine dello schermo 53
Bottoni della barra delle applicazioni 54
Registrazione dello schermo 54
Uso dello scanner 55
Avvio manuale del sistema grafico 55
Autorizzazione per l'accesso al servente X 55
Altre annotazioni 55
Accesso ad altri file system con NLNX 57
Innesto di altri file system 57
Accesso semplificato alle unità rimovibili 57
Accesso a unità non rimovibili 58
Limitazione all'accesso 58
Accesso a risorse di rete SMB 59
Sintesi dei comandi 61
Sintesi delle opzioni di avvio 65
Preparazione delle partizioni 69
Piano di utilizzo delle partizioni 69

Sistema di emergenza: «rescue» 70
Suddivisione in partizioni 70
Inizializzazione delle partizioni 73
Copia del sistema operativo 75
Copia di NLNX in una partizione di un disco «normale» ... 75
Copia in una directory 75
Configurazione 76
Configurazione alternativa 77
Scambio della memoria utilizzando un file 77
Installazione in un file-immagine 79
Installazione di un file-immagine in sola lettura 79
Installazione in sola lettura condivisa attraverso la rete 79
File-immagine in lettura e scrittura 80
Predisposizione del sistema di avvio 81
SYSLINUX 81
EXTLINUX 81
PXELINUX 82
Osservazioni sugli insiemi RAID 82
Osservazioni sulla convivenza di più sistemi operativi 82
Osservazioni sull'aggiornamento del kernel 83
Avvio di NLNX dopo la sua installazione 85
Configurazione dell'avvio 85
Cenni alla configurazione successiva 86
Realizzazione di una propria variante di NLNX 87
Modifica dei pacchetti applicativi installati 87
Creazione del file-immagine della versione *live* 87
Ricompilazione del kernel 91
Installazione di software non libero 92
Configurazione di NLNX per l'interazione con il sistema 93
Configurazione locale 93
Configurazione di mouse e tastiera 95
Configurazione di X 96
Stampa 99
Configurazione 99
Opzioni di avvio per la stampa 102
Particolarità varie di NLNX 103
ACPI 103
Configurazione di Bash e script di sistema 104
Procedura di inizializzazione del sistema 104
Registri 106
Pianificazione dei processi con Cron 106
Dati variabili nel DVD *live* 107
Blocco delle funzioni di «nlnxrc» 107
Servizi di rete vari, secondo l'organizzazione di NLNX 109
Nomi a dominio 109
DHCP 109
TFTP e PXE 110
HTTP 110
Proxy HTTP 110
File di registrazioni 111
Orologio di riferimento 111
Servizi da avviare manualmente 111
Utilizzo dello scanner 112
PXE per l'avvio di un elaboratore senza disco fisso 113

PXE 113
NLNX in un file system di rete 114
Controllo dell'avvio di sistemi locali 115
HTTP e servizi collegati 117
Impedire la pubblicazione di file personali 117
Programmi CGI 117
PHP 117
Webalizer 118
MRTG 118
Accessibilità dei registri (log) 119
Documentazione interna 119
Esercitazioni con il linguaggio HTML 119
La rete e gli instradamenti con NLNX 121
Interfacce di rete senza fili 121
Individuazione delle interfacce di rete 122
Connessione in una rete locale tradizionale 122
Connessione in una rete locale via radio 124
Router per una rete locale 127
Router per una rete locale, attraverso un proxy 129
Indirizzi di rete da evitare assolutamente 132
VNC con «nlnxrc» 133
Organizzazione e funzionamento 133
Utilizzo di un elaboratore remoto con un pubblico passivo 134
Utilizzo di un elaboratore locale con un pubblico passivo 135
Utilizzo di un elaboratore remoto con un pubblico attivo 135
Utilizzo di un elaboratore locale, con un pubblico attivo 136
Utilizzo senza parola d'ordine 136
Utilizzo di VNC attraverso un tunnel SSH 137
Utenze e amministrazione con NLNX 139
Creazione, modifica e cancellazione delle utenze, in modo interattivo 140
Creazione, modifica e cancellazione delle utenze, in modo non interattivo 143
NIS e NFS per utilizzare altre utenze 144
Controllo dello spazio utilizzato 145
Controllo del numero di pagine stampabili e dell'origine delle stampe 147
Utenze speciali per l'amministrazione 148
Utenze condivise e configurazione automatica con NLNX 151
Condivisione delle utenze attraverso NIS, NFS e Samba 151
Utilizzo di SSHfs al posto di NFS 152
Gestione delle utenze 153
Samba 154
Servente DHCP 157
Controllo dell'accesso a servizi HTTP esterni 161
Situazione tipica di utilizzo 161
Controllo a gruppi 161
Controllo dell'accesso alla stampante 163
Mettere in comunicazione insegnanti e studenti 165
Organizzazione delle utenze 165
Funzionamento 165
Estendere il controllo delle utenze alla rete MS-Windows 167
Situazione di esempio 167
Profili personali 168
Script di avvio 169

Utenze per gli elaboratori MS-Windows 169
Configurazione di MS-Windows XP Professional: disabilitazione della connessione cifrata 169
Configurazione di MS-Windows XP Professional: associazione al dominio 171
Configurazione di MS-Windows 7: definizione di due voci nel registro di sistema 172
Configurazione di MS-Windows 7: associazione al dominio 174
Spegnimento del servente NLNX 176
Servente WINS 176
Riferimenti 177
Installazione indolore di NLNX in una rete di elaboratori MS-Windows 179
Sequenza di avvio 179
File system principale attraverso la rete 179
Configurazione del DHCP 180
Configurazione dell'avvio 180
Riferimenti 181
Problemi di NLNX e soluzioni 183
Directory personale in un disco esterno 183
Utenze generiche 184
Messaggi di errore 184
Programmi «duri a morire» 184
Studenti troppo furbi 184
Vincolare gli utenti a un certo gruppo di postazioni 185
Adattamento di NLNX 187
Gerarchia doppia 187
Creazione di una nuova versione 188
Configurazione predefinita di NLNX 188
Aggiornamento dei pacchetti installati 189
Kernel 189
Organizzazione del laboratorio GNU/Linux 191
Accensione e spegnimento 192
Procedura per l'aggiunta di un utente 192
Controllo dei servizi 196
Controllo del numero massimo di pagine stampabili per volta 197
Utilizzo di elaboratori estranei al laboratorio 197
Utilizzo del laboratorio GNU/Linux 199
Avvio e arresto degli elaboratori del laboratorio 199
Utenze 200
Registri 200
Console e utilizzo di applicazioni grafiche 200
Stampa 201
Accesso a Internet 201
Licenze 202
Organizzazione dei laboratori MS-Windows 205
Procedura per l'aggiunta di un utente 206
Procedura per la modifica di una parola d'ordine 210
Controllo dell'accesso all'esterno 212
Configurazione di MS-Windows XP Professional per utilizzare le utenze centralizzate 212
Configurazione di MS-Windows 7 per utilizzare le utenze centralizzate 213
Configurazione per poter utilizzare anche un sistema GNU/Linux 217

Eliminazione delle utenze 218
Utilizzo dei laboratori MS-Windows 219
Premessa 219
Autenticazione 220
Accesso manuale ai dati personali da una postazione MS-Windows anonima 220
Scambio di dati tra insegnanti e studenti 222
Avvio di un sistema GNU/Linux 223
Registri elettronici 223
Servizio telnet.informaticalibera.net 225
Tutori, pupilli e directory personali 225
La stampa 226
Pubblicazione di file attraverso la directory «~/public_html/» 226
Elenco dei tutori e dei pupilli 227
Registri 227
Amministrazione da parte dei tutori 227
Accesso normale al sistema 229
Trasferimento dati tramite il protocollo SSH 230
Come funziona il servizio telnet.informaticalibera.net 231
Elaboratori coinvolti 231
Ridirezione del traffico TCP 231
Avvio dalla rete dell'elaboratore «B» 232
Dati personali delle utenze privilegiate 233
Servizi via HTTP presso l'elaboratore «A» 234
Riferimenti 235

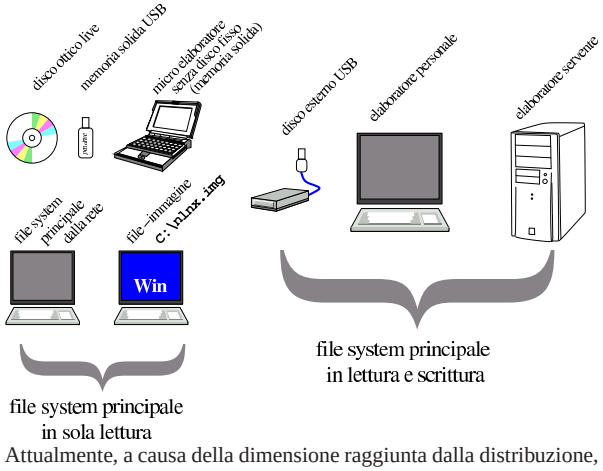
Semplicità e controllo

Caratteristiche generali 33
Insieme dei pacchetti applicativi disponibili 34
File system compresso 34
Espulsione del disco 34
Aggiornamento 34
Sistema di emergenza 34
Installazione: normale o in sola lettura 34
Avvio dalla rete 35

NLNX,[1] ex nanoLinux, è un sistema GNU/Linux versatile, per architettura x86, ottenuto a partire da pacchetti Debian. NLNX può funzionare sia in un file system in sola lettura (come quello di un disco ottico *live* o di una memoria solida USB), sia in un file system comune (come quello di un'unità a disco tradizionale). Inoltre, NLNX può essere installato in un file-immagine all'interno di un file system estraneo.

NLNX nasce e si evolve sulla base di esigenze legate alla gestione di reti di elaboratori destinati alla didattica; pertanto, NLNX deve risolvere problemi di gestione e di amministrazione relativi all'uso tipico che si fa di tali strumenti nei laboratori delle scuole. A questo proposito, il sistema operativo, così organizzato, è perfettamente adatto ad altri contesti di utilizzo, anche quando certe esigenze non sono così pressanti come invece avviene a scuola.

Figura u11.1. Campi principali di applicazione di NLNX.



Attualmente, a causa della dimensione raggiunta dalla distribuzione, non è più possibile inserire questa in un DVD-ROM, pertanto è anche temporaneamente sospeso l'invio gratuito della stessa, in attesa che si diffondano i dischi BD (*Blu-ray disc*).

Caratteristiche generali

Il motto di NLNX è «semplicità e controllo», in riferimento all'organizzazione e gestione del sistema operativo, nel senso che **una sola persona** dovrebbe poter gestire, **nei ritagli di tempo**, una rete numerosa di elaboratori facenti capo a un solo servente, basando tutto su NLNX. Pertanto, l'aggettivo «semplicità» non va frainteso, in quanto servono comunque delle competenze, ma almeno la filosofia di NLNX è quella di evitare complicazioni dove possibile.

NLNX può essere modificato e riprodotto in forma di disco ottico *live* o di unità di memoria solida USB: basta installarlo in un elaboratore tradizionale e avere a disposizione abbastanza spazio libero per i file temporanei che si generano con tale procedimento.

Per quanto riguarda la gestione della rete, l'organizzazione di NLNX è rivolta a un contesto in cui l'accesso alla rete esterna avviene tramite un router già presente e accessibile dalla rete locale.

Dal punto di vista della grafica, NLNX è privo di «effetti speciali»; in particolare non si usa nemmeno un gestore di sessione e non c'è una preferenza per Gnome o KDE, in quanto si usano indifferentemente programmi di entrambi gli ambienti. La gestione dell'audio è essenziale e di norma un solo programma per volta può disporre delle funzionalità audio.

NLNX offre potenzialmente dei servizi impegnativi anche quando viene avviato da un'unità in sola lettura (come i dischi ottici *live* e le unità USB), salvo qualche eccezione per motivi di sicurezza.

Insieme dei pacchetti applicativi disponibili

L'edizione standard attuale di NLNX, ha un insieme di programmi applicativi scelti secondo criteri di funzionalità, cercando di evitare ridondanze dove possibile. Nella scelta dei programmi sono considerati la snellezza, la capacità di funzionare correttamente con la maggior parte delle lingue gestibili.

In generale, si preferisce la «pratica» rispetto alla «teoria». Ovvero, anche se sarebbe preferibile un certo programma ***x***, può darsi che in pratica quel programma manifesti qualche inconveniente nelle condizioni di uso a cui è destinato NLNX; pertanto si opta per un programma ***y*** che magari è meno efficiente o più pesante, ma che almeno sembra dare, in pratica, maggiori garanzie di affidabilità.

Per la gestione della stampa si utilizza LPRng e non ci si avvale di CUPS. Per problemi di dipendenze tra i pacchetti applicativi, ne esistono alcuni di fittizi, senza alcun contenuto.

File system compresso

Nelle edizioni attuali di NLNX, a eccezione di alcuni file necessari per l'avvio, il file system del disco ottico *live* contiene dati compressi. Per la precisione, i dati compressi sono contenuti in un file system Squashfs (sezione 19.9), rappresentato dal file immagine '`nlnx.img`'.

Espulsione del disco

Nelle edizioni attuali di NLNX, quando il sistema viene utilizzato direttamente da disco ottico *live*, al termine del suo funzionamento tenta di espellere automaticamente il disco.

Aggiornamento

NLNX non dispone di una procedura di aggiornamento. Se si vuole «aggiornare» un sistema installato nel modo tradizionale, occorre fare una copia di quello vecchio, reinstallare quello nuovo e quindi ripristinare manualmente la configurazione precedente, assieme a tutti gli altri dati che prima venivano gestiti nello stesso file system.

Sistema di emergenza

NLNX si avvale di un disco RAM iniziale per l'avvio. Questo disco RAM include un proprio sistema minimo che generalmente mostra un menù di funzioni, per lo più rivolte alla selezione dell'unità da usare per l'avvio. Ma nel menù è prevista anche la voce *rescue*, con la quale si ottiene un sistema minimo di emergenza, da usare per la manutenzione. Questo sistema minimo può consentire l'accesso a directory condivise in rete attraverso il protocollo NFS ed eventualmente è in grado di configurarsi automaticamente se è disponibile un servizio DHCP.

Installazione: normale o in sola lettura

L'installazione ottimale di NLNX prevede l'uso esclusivo di una partizione di un disco fisso, tale da consentire un utilizzo «normale». Al contrario, il funzionamento da un disco ottico *live* o da unità di memoria solida USB (o anche da altri contesti), è differente, in quanto, in tal caso il file system è in sola lettura.

Teoricamente, si potrebbe installare NLNX in una memoria solida USB, come se si trattasse di un disco fisso comune. Tuttavia, gestendo un file system in lettura e scrittura, si hanno due tipi di problemi: le operazioni di scrittura sono molto lente e tutte le operazioni che si svolgono ne sarebbero coinvolte, compreso l'avvio che diventerebbe lentissimo, ma soprattutto, **la scrittura continua distruggerebbe rapidamente l'unità di memoria**.

Avvio dalla rete

Con NLNX è possibile allestire un servente in grado di gestire l'avvio di altri sistemi NLNX attraverso la rete. Ciò può comportare due tipi di approccio: l'avvio di un sistema remoto, non disponendo localmente di una memoria di massa adeguata, oppure l'avvio di un sistema installato localmente, per il quale non è però necessario intervenire modificando il settore di avvio.

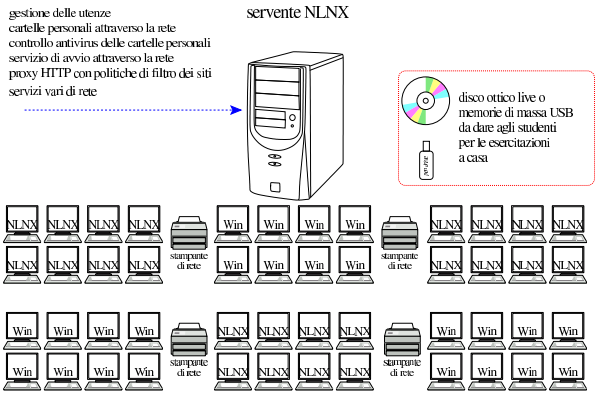
[1] **NLNX** GNU GPL; i singoli applicativi sono sottoposti eventualmente a proprie condizioni specifiche

Ambiente operativo ideale di NLNX

Servente 37
Elaboratori periferici 37
Accesso alle stampanti 38
Proxy HTTP 38
Il lavoro a casa 38
Osservazioni 38

NLNX è organizzato per soddisfare delle esigenze che vanno dalla realizzazione di dischi ottici *live* fino all'amministrazione di una rete locale importante, con la centralizzazione delle utenze e dei dati personali (cartelle o directory personali). L'ambiente di riferimento è la scuola (escludendo però la parte amministrativa, per la quale è necessario ci sia una separazione fisica, rispetto alla rete didattica), dove la quantità di utenti e di postazioni di lavoro è sempre molto elevata.

Con NLNX è possibile amministrare le utenze, sia per altri sistemi NLNX, sia per sistemi MS-Windows, ma soprattutto è possibile **consentire** l'uso di un sistema GNU/Linux a chi lo voglia, senza essere per questo costretti a predisporre doppie partizioni e sistemi di avvio multipli nella stessa memoria di massa.



Servente

Tutto quello che serve per la gestione dei servizi legati alla rete, viene collocato all'interno di un solo elaboratore, provvedendo però ad assicurare un sistema di copie di sicurezza, tale da permettere un ripristino tempestivo, anche in un altro elaboratore, in caso di avaria.

Per il servente è sufficiente hardware comune e una sola unità a disco di capacità media. Ciò su cui conviene puntare è la qualità della CPU e la quantità di memoria centrale. In particolare, una quantità elevata di memoria centrale serve ad alleggerire il carico di lavoro dell'unità a disco.

Elaboratori periferici

Gli elaboratori periferici che si avvalgono del servente per la gestione delle utenze e per gli altri servizi, possono funzionare indifferentemente con un sistema NLNX o con MS-Windows, fornendo le stesse utenze e gli stessi dati personali agli utenti. Inoltre, gli elaboratori che vanno utilizzati normalmente con MS-Windows possono essere avviati, attraverso la rete, mettendo in funzione un sistema NLNX equivalente a quello degli elaboratori che ne hanno uno installato stabilmente. Il sistema NLNX offerto attraverso la rete risiede materialmente nel servente e la sua configurazione viene gestita attraverso opzioni di avvio.

Accesso alle stampanti

Le stampanti accessibili attraverso la rete, possono essere stampanti di rete vere e proprie, stampanti offerte da elaboratori NLNX, oppure stampanti condivise attraverso elaboratori MS-Windows. Gli elaboratori NLNX sono in grado di stampare attraverso stampanti condivise di MS-Windows, oltre che da stampanti di rete vere e proprie; inoltre, tale configurazione è gestibile anche quando NLNX viene avviato attraverso la rete.

Proxy HTTP

Per controllare l'utilizzo della rete, relativo al protocollo HTTP, gli elaboratori periferici possono essere configurati in modo da sfruttare il servente NLNX come router per l'accesso all'esterno, il quale va però predisposto per la gestione del servizio proxy trasparente. Il servizio proxy è gestito in modo molto semplice, per evitare conflitti con MS-Windows; per esempio è assente il controllo preventivo dei contenuti.[1]

L'utilizzo del servizio proxy, in modo trasparente come accennato, oltre che consentire un filtro di massima a indirizzi o siti impropri, permette di controllare l'accessibilità o meno delle risorse esterne alle singole postazioni, di volta in volta, anche a utenti che abbiano ottenuto delle facoltà limitate di amministrazione.

Il lavoro a casa

Una copia di NLNX può essere data agli studenti, trattandosi esclusivamente di software libero, in un disco ottico *live* o in un'unità di memoria solida USB. In tal modo gli studenti possono svolgere a casa le stesse esercitazioni, senza bisogno di avere le competenze necessarie a installare effettivamente un sistema GNU/Linux.

Osservazioni

NLNX non ha lo scopo di appoggiare o di assecondare l'uso di software proprietario, ma rimane il fatto che, in molte scuole italiane, sia pressocché «obbligatorio»[2] disporre di laboratori didattici basati sul sistema operativo MS-Windows. Di fronte a questa situazione, per garantire la libertà di insegnamento di chi crede fermamente nel valore del software libero, NLNX offre una soluzione che consente a tutti di rimanere nelle proprie posizioni, senza richiedere risorse eccessive e senza fare discriminazioni.

[1] Originariamente, NLNX includeva Dansguardian, ovvero un sistema abbastanza sofisticato per il controllo dei contenuti. Attualmente c'è solo OOPS, con il quale il controllo è meno preciso, ma rimane almeno la possibilità di eliminare selettivamente gli indirizzi che si rivelano inadatti al contesto di utilizzo.

[2] Questo stato di fatto si scontra sostanzialmente con la direttiva «Stanca» che viene citata in fondo al testo.

NLNX per il principiante

Utilizzo del DVD *live* 39
Installazione in «C:\» 39
Utenze 40
Configurazione automatica della rete 41
Voci per l'avvio 41

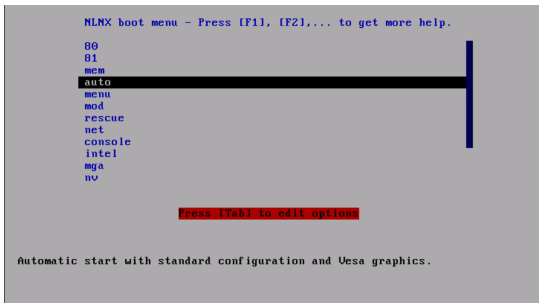
Il principiante può utilizzare NLNX in sola lettura; in tal modo, il riavvio o lo spegnimento accidentale dell'elaboratore non dovrebbero produrre effetti indesiderabili nell'ambito del sistema MS-Windows o di qualunque altro sistema ospitante. Naturalmente, se NLNX viene usato per produrre dei file, questi vanno salvati in unità rimovibili esterne.

Utilizzo del DVD *live*

NLNX è un sistema GNU/Linux che può essere avviato da un disco ottico, senza bisogno di installazione. Per fare questo è necessario che il BIOS sia configurato in modo tale da avviare per primo quanto contenuto nell'unità per dischi ottici.

```
First  boot device      CDROM/DVDROM
Second boot device      Floppy
Third  boot device      HDD-0
Fourth boot device      none
```

Quando si avvia il disco ottico, si ottiene una schermata simile a quella successiva:



Se non si fa nulla, dopo un po' si avvia NLNX con una configurazione predefinita; diversamente si può scegliere una voce specifica dal menù, premendo poi [*Invio*] per conferma.

La distribuzione di NLNX prevede una configurazione automatica dello schermo grafico; ma, in alcuni casi, non funziona. Per quelle situazioni particolari, NLNX può essere avviato scegliendo una voce diversa dal menù iniziale, oppure specificando opzioni particolari, per le quali, però, si richiede un approfondimento del sistema operativo. Se si hanno problemi con la grafica che non si risolvono scegliendo una voce specifica dal menù di avvio iniziale, si può scegliere la voce *console* (al posto di *auto*), ma in tal caso la grafica non viene avviata affatto.

Installazione in «C:\»

In alternativa all'utilizzo da un disco ottico *live*, NLNX può essere installato in modo molto semplice, copiando il file-immagine '`nlnx.img`' nella cartella '`C:\`' di un sistema MS-Windows, senza provocare traumi al sistema ospitante.

Dopo la copia, occorre trovare un modo per avviare NLNX, al posto di MS-Windows. Inizialmente, conviene configurare il BIOS in modo da avviare prima dall'unità per dischi ottici, rispetto al disco fisso normale. In questo modo, quando si vuole usare NLNX, ci si può avvalere del disco di NLNX.

Quando il disco ottico viene avviato, selezionando la voce *menu*, si ottiene un elenco in cui appaiono due voci appartenenti normalmente alla stessa versione di NLNX:

```
.----------------NLNX initial ram disk----------------.
| Please select the root file system.                 |
| .-------------------------------------------------. |
| |   sda1     NLNX_2010.02.21_img_ro               | |
| |   hda      NLNX_2010.02.21_img_ro               | |
| |   rescue   start a rescue system                | |
| |   modules  a shell to load modules manually     | |
| |   net boot boot a network, read only, NLNX      | |
| `-------------------------------------------------' |
|-----------------------------------------------------|
|             <  OK  >       <Cancel>                |
`-----------------------------------------------------'
```

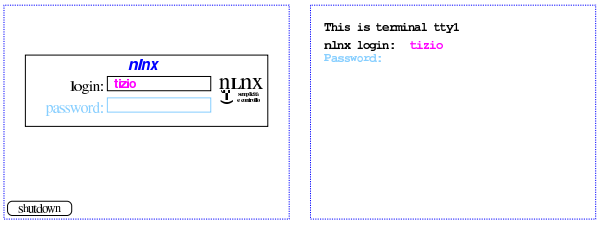
In generale, la prima delle due voci dovrebbe essere quella corretta. Si può provare a selezionarla, portandovi sopra la barra di selezione e premendo poi [*Invio*]. Se durante il processo di avvio si vede un'attività intensa del disco ottico, significa che si tratta invece della voce sbagliata; occorre quindi riavviare e scegliere la seconda.

Utenze

Per utilizzare il sistema operativo è necessario che l'utente si identifichi, attraverso l'indicazione di un nominativo-utente e di una parola d'ordine. Quando si usa il file system in sola lettura, in mancanza d'altro sono disponibili delle utenze predefinite: '**`tizio`**', '**`caio`**', '**`sempronio`**',... In ogni caso la parola d'ordine per l'identificazione è sempre «nlnx» (solo lettere minuscole).

A seconda dei casi, l'identificazione può essere richiesta in forma grafica o meno, ma in entrambe le situazioni, la parola d'ordine che viene inserita non appare sullo schermo, nemmeno in forma di asterischi o di pallini; in altri termini, si tratta di una **digitazione che avviene completamente all'oscuro**.

Figura u13.4. *Login*: a sinistra nella modalità grafica; a destra in una console tradizionale.



Eccezionalmente, è possibile configurare NLNX all'avvio, in modo da essere utilizzato esclusivamente con le utenze predefinite, senza bisogno di inserire una parola d'ordine per l'autenticazione. In tal caso, si presenta un menù simile a quello della figura successiva, dove basta selezionare l'utente prescelto, per ottenere poi un funzionamento in modalità grafica.

Figura u13.5. Menù per la selezione di un'utenza predefinita, senza bisogno di fornire poi la parola d'ordine.

```
.------Default user selection--------.
| Please select a default user:      |
| .--------------------------------. |
| |   tizio      /home/tizio       | |
| |   caio       /home/caio        | |
| |   sempronio  /home/sempronio   | |
| |   mevio      /home/mevio       | |
| |   filano     /home/filano      | |
| |   martino    /home/martino     | |
| |   calpurnio  /home/calpurnio   | |
| |   ---------  ---------------   | |
| |   reboot     system reboot     | |
| |   shutdown   system shutdown   | |
| `------------------------100%----' |
|                                    |
|------------------------------------|
|       <  OK   >    <Annulla>       |
`------------------------------------'
```


Si possono utilizzare sei console virtuali, oltre alla sessione grafica. Per passare da una sessione all'altra si usa la combinazione di tasti [*Ctrl Alt F**n***]. Per la precisione si tratta di [*Ctrl Alt F1*], [*Ctrl Alt F2*],... fino a [*Ctrl Alt F7*] ed eventualmente [*Ctrl Alt F8*]: le prime sei riguardano le console virtuali; la settima combinazione riguarda la sessione grafica e l'ottava, eventuale, riguarda il menù dell'ultima figura mostrata.

Figura u13.6. Selezione delle sessioni di lavoro: a sinistra le combinazioni di tasti per le console virtuali; a destra la combinazione per la sessione grafica.

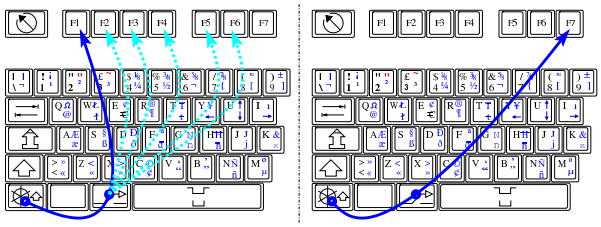
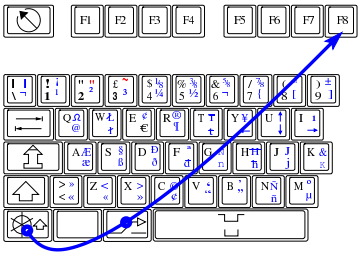


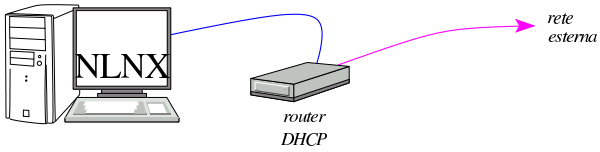
Figura u13.7. Selezione della console virtuale che ospita il menù, quando è attiva la modalità di accesso alle sole utenze predefinite, senza bisogno di parola d'ordine.



Configurazione automatica della rete

Per poter utilizzare NLNX in rete, senza disporre di competenze particolari al riguardo, è necessario un router configurato per il servizio DHCP. In tal modo, NLNX assume automaticamente la configurazione da tale router, al momento dell'avvio.

Figura u13.8. Router che offre il servizio DHCP e consente la configurazione automatica della rete all'avvio di NLNX.



Voci per l'avvio

Al momento dell'avvio, oltre a '**`auto`**' o '**`console`**', è possibile selezionare molte altre voci, a cui corrispondono comportamenti e configurazioni differenti di NLNX. La tabella successiva descrive quelle principali.

Tabella u13.9. Sigle principali per la selezione della modalità di avvio del disco ottico *live* e di unità esterne USB.

Sigla	Comportamento
`80` `81`	Rinvia al primo o al secondo disco locale, ammesso che disponga di un settore MBR con il codice di avvio.
`mem`	Avvia il programma Memtest86+ (descritto nella sezione 9.10) per l'analisi dello stato della memoria centrale.
`auto`	Avvia automaticamente NLNX, utilizzando una configurazione generale predefinita.
`menu`	Avvia NLNX, sospendendo il processo di avvio con un menù per la scelta finale del file system principale o di altre opzioni.

Sigla	Comportamento
`rescue`	Avvia il kernel di NLNX e il disco RAM iniziale, limitandosi a offrire un sistema minimo di emergenza.
`net`	Avvia NLNX innestando un file system di rete, ammesso che sia disponibile il servente per questo tipo di servizio.
`console`	Avvia NLNX, sospendendo il processo di avvio con un menù per la scelta finale del file system principale. A differenza della voce '**`menu`**', non avvia la grafica.

`intel`
`mga`
`nv`
`radeon`
`vesa`
	Avvia NLNX, sospendendo il processo di avvio con un menù per la scelta finale del file system principale. A differenza della voce '**`menu`**', richiede una configurazione grafica particolare, corrispondente al nome usato per la voce stessa.

`hdx` `sdx`	Si tratta di una voce da modificare al volo, con la quale si avvia NLNX utilizzando un file system contenuto del file di dispositivo '`/dev/hd`…' o '`/dev/sd`…'.

La configurazione grafica di NLNX

Menù 43
Configurazione delle finestre 44
Comandi da tastiera 46
Configurazione locale 47
Configurazione della mappa della tastiera 47
Livelli di inserimento 48
Accenti morti 48
Metodi di inserimento 50
Messaggi di errore 50
Avvio dei programmi 51
Chiusura dei programmi 51
Configurazione del mouse 53
Sfondo 53
Cattura dell'immagine dello schermo 53
Bottoni della barra delle applicazioni 54
Registrazione dello schermo 54
Uso dello scanner 55
Avvio manuale del sistema grafico 55
Autorizzazione per l'accesso al servente X 55
Altre annotazioni 55

`.Trash/` 53 `.wallpaper` 53 `Desktop/` 53

NLNX dispone di un servente X per l'uso di applicazioni grafiche, assieme al gestore di finestre Fvwm, ma senza un gestore di sessione.

Menù

Per ottenere il menù delle applicazioni, va selezionato il primo bottone grafico della barra che si solleva quando vi si porta sopra il puntatore del mouse.

Figura u14.1. La barra delle applicazioni.

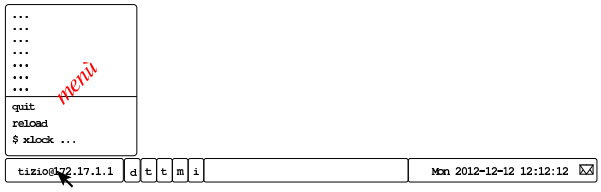
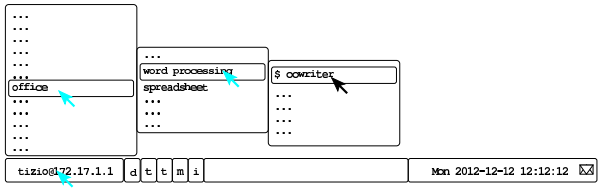


Figura u14.2. Avvio di OpenOffice.org Writer.

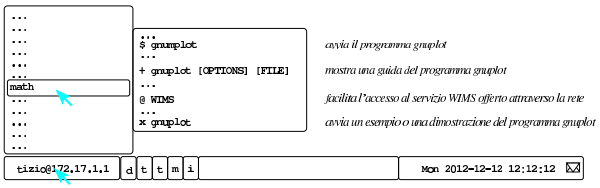


Le voci del menù possono avere dei simboli che inizialmente appaiono oscuri, ma che invece hanno un significato preciso. Le voci che iniziano con un dollaro ('**$**'), oppure con un cancelletto ('**#**') rappresentano l'inizio di un comando, così come presumibilmente andrebbe inserito attraverso una finestra di terminale. Che appaia un dollaro o un cancelletto, dipende dal fatto che si stia usando il sistema in qualità di utente comune oppure come amministratore, ma in certi casi il cancelletto anche se si agisce come utente comune, denotando la necessità di ottenere i privilegi dell'amministratore per poter eseguire il comando stesso. Nel caso dell'ultima figura, la voce

‘**`$ oowriter`**’ indica che la sua selezione è equivalente a eseguire il comando:

```
$ oowriter [Invio]
```

Figura u14.3. Simboli usati nelle voci del menù.



Il simbolo ‘**`+`**’ indica una voce dalla quale ottenere una guida all’uso del programma; pertanto, ‘**`+ gnuplot`**’ serve a ottenere le istruzioni per usare il programma omonimo. Il simbolo ‘**`x`**’ indica una voce da cui si ottiene un esempio o una dimostrazione del funzionamento del programma che segue. Il simbolo ‘`@`’ indica una voce riferita a un servizio offerto attraverso la rete.

Per chiudere una sessione di lavoro, si seleziona dal menù la voce *quit* e quindi si conferma ancora con *quit*. Eventualmente, se la procedura normale per chiudere la sessione di lavoro non dovesse funzionare, si può utilizzare la combinazione di tasti [*Ctrl Alt Backspace*] (ovvero [*Ctrl Alt* < - - -]).

Figura u14.4. Chiusura della sessione di lavoro dell’utente.

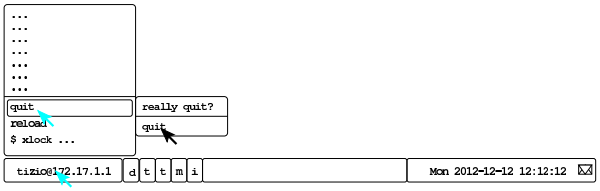
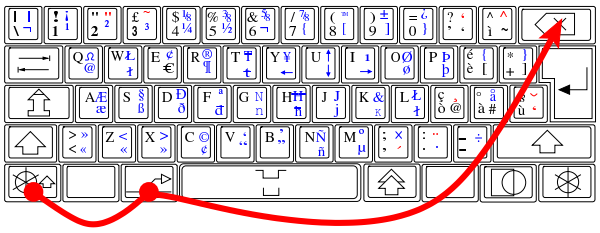


Figura u14.5. Uso della combinazione di tasti [*Ctrl Alt Backspace*].

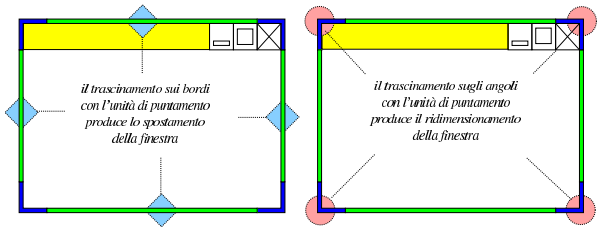


La barra delle applicazioni, da dove si apre il menù, potrebbe scomparire, quando viene eliminato involontariamente il suo processo elaborativo. In tal caso, il menù si ottiene premendo il tasto destro del mouse sulla superficie vuota dello sfondo grafico, e da lì si può selezionare la voce *reload window manager*. Questa operazione non interferisce con le altre applicazioni già in funzione.

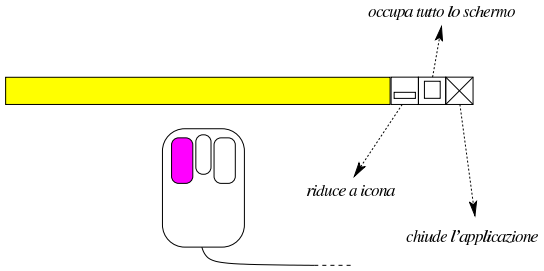
Figura u14.6. Riavvio del gestore di finestre.

Configurazione delle finestre

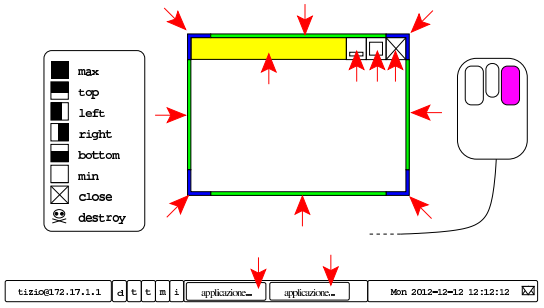
L’uso delle applicazioni grafiche, secondo la configurazione del gestore di finestre, è abbastanza intuitivo, ma ci sono degli accorgimenti inusuali che possono essere interessanti. Il trascinamento con il puntatore del mouse sui bordi delle finestre produce lo spostamento o il ridimensionamento delle stesse, a seconda che si trovi, rispettivamente, sulla parte rettilinea o sugli angoli.



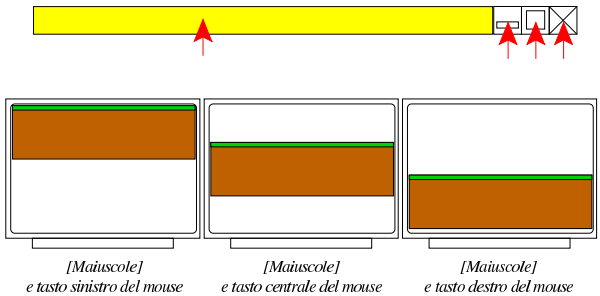
I bottoni che appaiono sulla barra superiore della finestra, se selezionati con il primo tasto del mouse (o dell’unità di puntamento), si comportano secondo la modalità consueta:

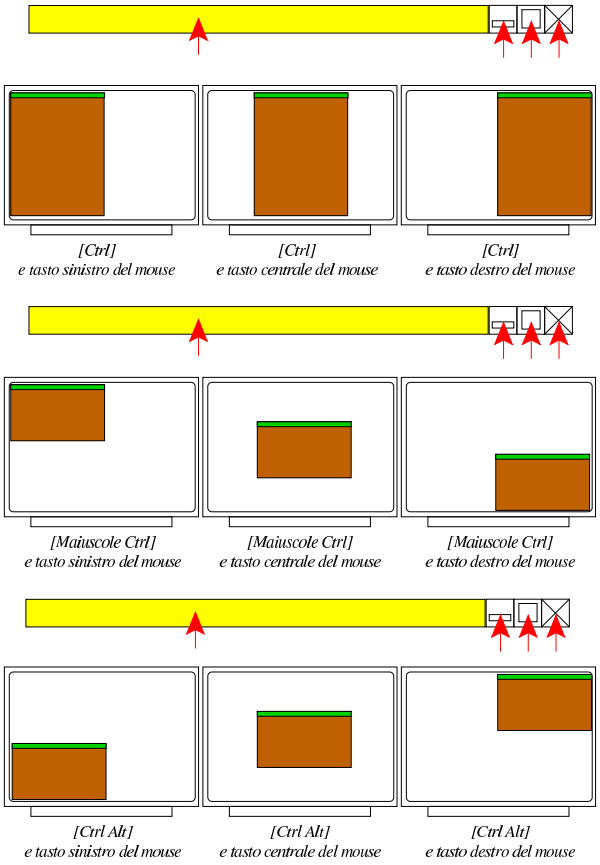


Se invece si usa il terzo tasto del mouse (generalmente il destro), su qualunque superficie esterna della finestra, inclusa la barra del titolo, i bottoni già visti e i bottoni delle applicazioni, si ottiene un menù per il controllo della dimensione della finestra o del processo.



Se si combinano i tasti [*Maiuscole*], [*Ctrl*] e [*Alt*], quando si seleziona la barra del titolo o gli altri bottoni di una finestra, si ottiene un ridimensionamento particolare della stessa:

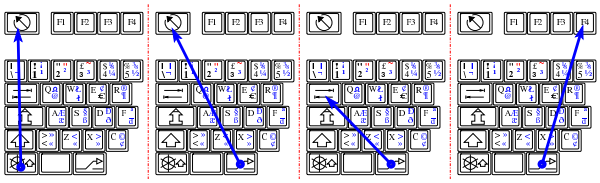




Comandi da tastiera

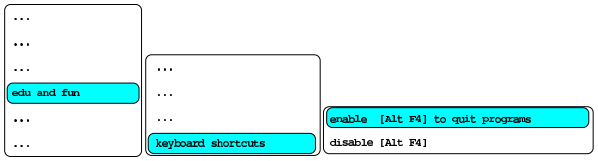
Sono disponibili alcuni comandi che possono essere impartiti attraverso la tastiera, che si comportano in modo simile a quello di altri sistemi operativi. Lo specchietto successivo riepiloga le combinazioni possibili, ma occorre sottolineare che la combinazione [*Alt F4*] funziona solo se viene abilitata espressamente dall'utente.

Combinazione	Effetto
[*Ctrl Esc*]	Fa apparire il menù principale, che può essere attraversato con l'aiuto dei tasti freccia e per selezionare la voce evidenziata basta premere il tasto [*Invio*].
[*Alt Esc*]	Mette in primo piano la prossima finestra; ripetendo la combinazione si scorre tra le applicazioni.
[*Alt Tab*]	Fa apparire un menù con l'elenco delle applicazioni disponibili, dove è possibile selezionare quella desiderata per portarla in primo piano.
[*Alt F4*]	Se abilitato, chiude l'applicazione attiva o conclude la sessione grafica, ma solo dopo una conferma.



L'abilitazione o la disabilitazione delle combinazione [*Alt F4*] è accessibile dal menù di Fvwm, come si vede nella figura successiva.

Figura u14.16. Abilitazione o disabilitazione della combinazione [*Alt F4*], attraverso le voci del menù di Fvwm.



Configurazione locale

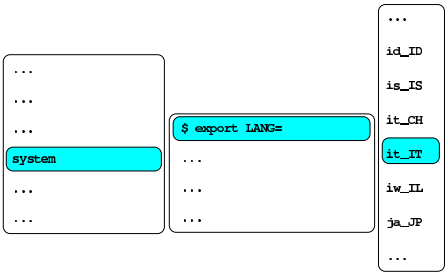
In condizioni normali, in un sistema GNU/Linux si utilizza la variabile '**LANG**' o le variabili '**LC_***' per dichiarare il linguaggio preferito e le convenzioni locali. La scelta di un linguaggio implica anche quella di una codifica appropriata per la rappresentazione dei caratteri a video e per la scrittura dei file di testo, ma dal momento che la console di un sistema GNU/Linux è in grado di gestire un numero limitato di caratteri per lo schermo, NLNX adotta uno strattagemma.

NLNX introduce l'uso della variabile di ambiente '**LANG_FOR_X**', con la quale si può stabilire una configurazione specifica per l'uso dell'ambiente grafico. Per esempio, si potrebbe gestire la console secondo le convenzioni statunitensi ('**en_US.UTF-8**'), mentre si può impostare la grafica per la lingua e le convenzioni della Russia ('**ru_RU.UTF-8**').

Si osservi che se si avvia una finestra di terminale con una shell di *login* (l'opzione '**-ls**' di '**xterm**' e di altri programmi simili fa questo), la configurazione che si ottiene in quell'ambito è la stessa della console.

Per modificare la configurazione locale durante il funzionamento in modalità grafica, si può accedere alla voce *system*, *$ export LANG=* e selezionare la combinazione di lingua e nazionalità preferita. Nell'elenco non si vede il nome della codifica, perché è stabilito implicitamente che si tratti di UTF-8.

Figura u14.17. Selezione della configurazione locale durante il funzionamento in modalità grafica.

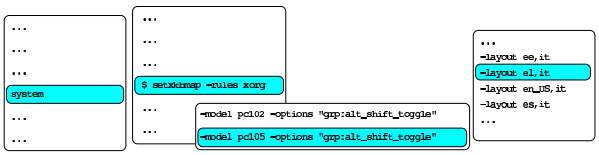


Configurazione della mappa della tastiera

Dal momento che NLNX prevede una configurazione locale adatta anche a lingue che si scrivono con un alfabeto non latino, quando si vuole configurare la mappa della tastiera per una lingua del genere, c'è poi il problema di scrivere comandi o direttive che invece richiedono l'alfabeto latino. Per ridurre gli inconvenienti dovuti a queste esigenze, la configurazione della mappa della tastiera con il sistema grafico, prevede sempre la selezione di una coppia: una mappa principale e una alternativa.

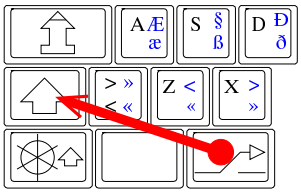
In base alla logica prevista, la mappa principale deve essere quella che consente di scrivere con l'alfabeto latino; probabilmente, se la propria lingua non lo prevede, la mappa principale potrebbe essere quella statunitense. Successivamente, durante il funzionamento è sempre possibile cambiare la mappa alternativa. Per esempio, se la mappa principale, già definita in fase di configurazione, è quella italiana ('**it**'), dal menù si può selezionare la sequenza di voci seguenti

per avere una mappa alternativa in greco:
system,
$ setxkbmap -rules xorg,
-model pc105 -options "" -options "grp:alt_shift_toggle",
-layout el,it.

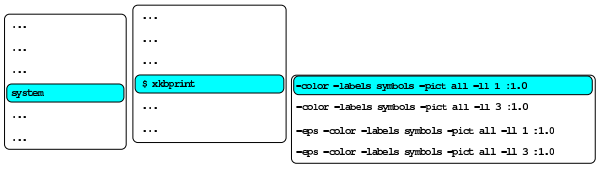



```
$ setxkbmap -rules xorg -model pc105 -options "grp:alt_shift_toggle" -layout el,it
```

Per passare dalla mappa principale a quella alternativa e viceversa, è sufficiente la combinazione di tasti [*Alt Maiuscole*], oppure, si possono anche «scambiare» le mappe con la voce *keymap groups* del menù principale.



Si può ottenere la visualizzazione della mappa attiva principale, procedendo così:
system,
$ xkbprint,
*-color -labels symbols -pict all -ll 1 :**n***
(oppure *-color -labels symbols -pict all -ll 3 :**n*** per le funzioni del terzo e quarto livello della tastiera).

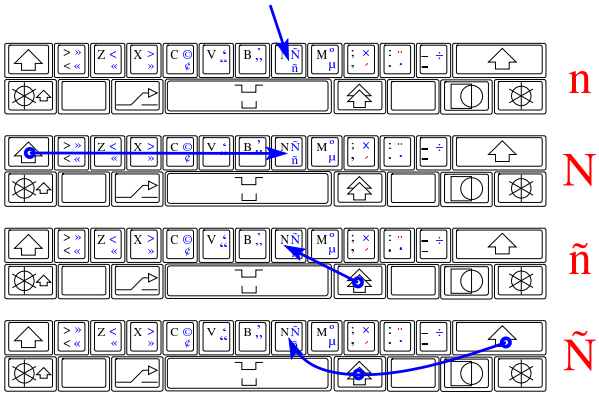



```
$ xkbprint -color -labels symbols -pict all -ll 1 :1.0
```

Livelli di inserimento

La maggior parte delle mappe prevede l'uso del tasto [*AltGr*] per passare al terzo livello (per esempio, quello che nella disposizione italiana consente di ottenere la chiocciola, il cancelletto, le parentesi quadre e il simbolo dell'euro). Si veda anche quanto scritto a partire dalla sezione 28.6 sulla configurazione e sull'uso della tastiera con X.

Figura u14.21. Livelli della tastiera: in questo caso si tratta della mappa italiana e si vede la selezione dei quattro livelli in corrispondenza del tasto [*n*].



Accenti morti

Diverse mappe prevedono degli «accenti morti», ovvero tasti o combinazioni di tasti che inizialmente non producono nulla, in attesa della pressione del carattere successivo. Per esempio, per ottenere il carattere «ô», si preme la combinazione [*Maiuscole AltGr* ^] e poi il tasto [*o*]. In altri termini, si seleziona prima l'accento morto e poi il carattere da accentare, ammesso che esista un carattere di quel tipo con quel accento particolare.

Figura u14.22. Accento circonflesso morto.

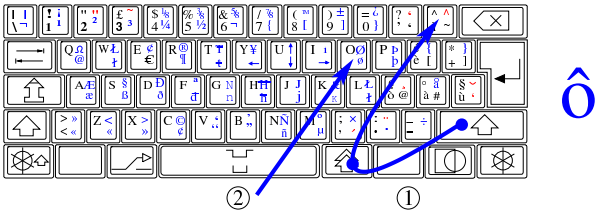


Figura u14.23. Accento acuto morto.

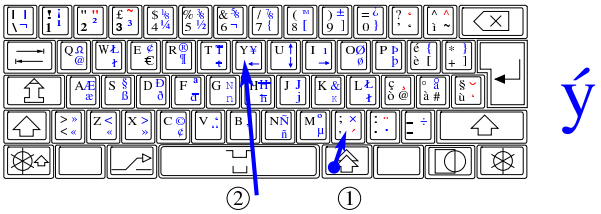


Figura u14.24. Accento grave morto.

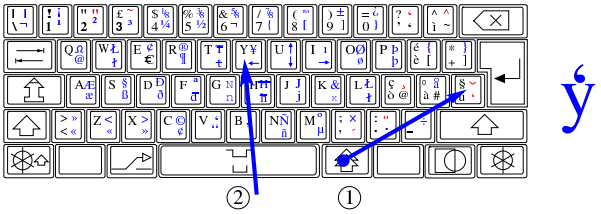


Figura u14.25. Dieresi.

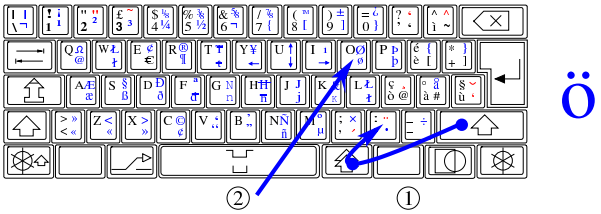


Figura u14.26. Tilde.

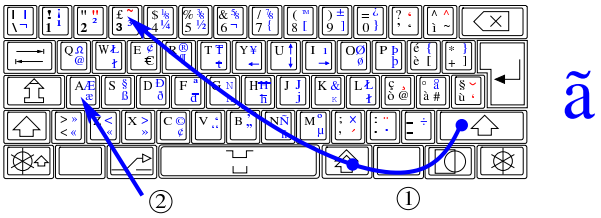
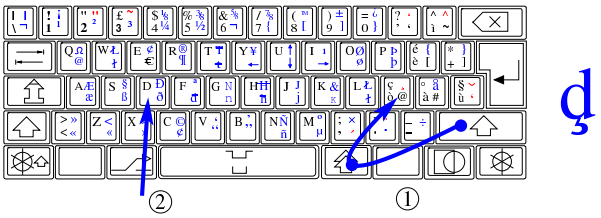


Figura u14.27. Cediglia.



Metodi di inserimento

NLNX consente di configurare al volo la mappa della tastiera, per tutte le lingue che prevedono un numero limitato di caratteri. Quando si ha a che fare con lingue asiatiche con molti caratteri, quale è il caso del cinese, occorre avvalersi di quello che è noto come «metodo di inserimento intelligente». D'altra parte, il fatto di non disporre di una tastiera con la rappresentazione dei simboli, anche quando questi sono in numero limitato, rende più semplice un metodo di inserimento basato su una forma di traslitterazione.

Nelle edizioni di NLNX standard che dispongono della grafica, all'avvio di X viene avviata anche una copia del demone '**`scim`**', per facilitare l'inserimento di un testo utilizzando i caratteri di varie lingue. Per attivare o disattivare questa funzione speciale, si utilizza la combinazione di tasti [*Ctrl Spazio*], quando è attiva l'applicazione con la quale si è in fase di scrittura. Tuttavia, la combinazione di tasti [*Ctrl Spazio*] non funziona sempre, perché alcuni programmi richiedono espressamente che sia stata predisposta una configurazione locale appropriata (per esempio '**`zh_CN.UTF-8`**' per il cinese).

Figura u14.28. Attivazione e sospensione del metodo di inserimento intelligente, attraverso la combinazione [*Ctrl Spazio*].

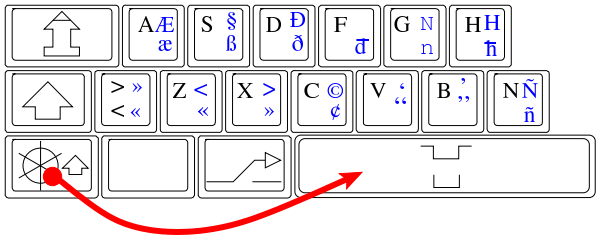
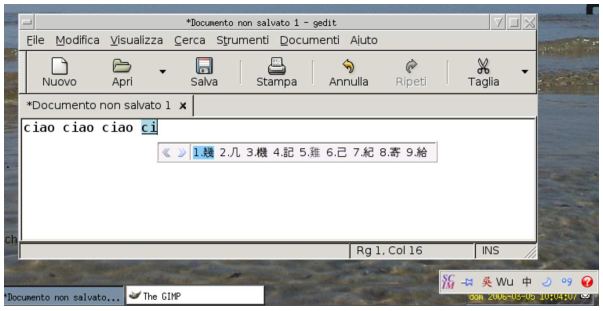


Figura u14.29. Attivazione del metodo di inserimento intelligente per la lingua cinese, durante l'uso di un programma per la modifica di file di testo.



È bene osservare che, qualunque metodo di inserimento sia stato scelto, per ottenere il risultato occorre che la configurazione della tastiera preveda l'uso dell'alfabeto latino; inoltre, è necessario che siano disponibili (installati) i caratteri necessari alla scrittura, altrimenti il testo non può essere visualizzato.

Messaggi di errore

Quando si avvia un programma, possono verificarsi dei problemi, che normalmente vengono segnalati attraverso lo standard error. Tuttavia, durante il funzionamento in modalità grafica si tendono a perdere tali messaggi, perché questi programmi sono avviati attraverso un menù e non un terminale comune. Per ovviare a questo inconveniente, la configurazione particolare di NLNX è organizzata in modo da mostrare lo standard output e lo standard error generato dal gestore di finestre, attraverso un terminale trasparente senza titolo, che si trova apparentemente sullo sfondo, la cui attivazione deve però essere richiesta espressamente:

standard output and error window,
new window.

Figura u14.30. Attivazione della finestra trasparente con i messaggi inviati dai programmi allo standard output e allo standard error.

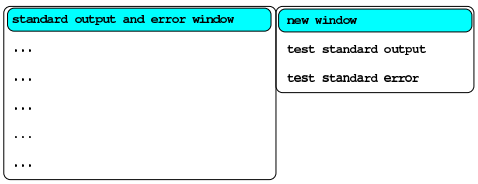
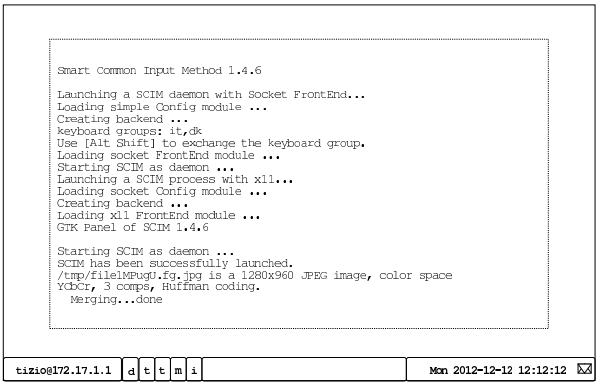


Figura u14.31. La finestra dei messaggi, dove si vedono alcune informazioni generate dai programmi.

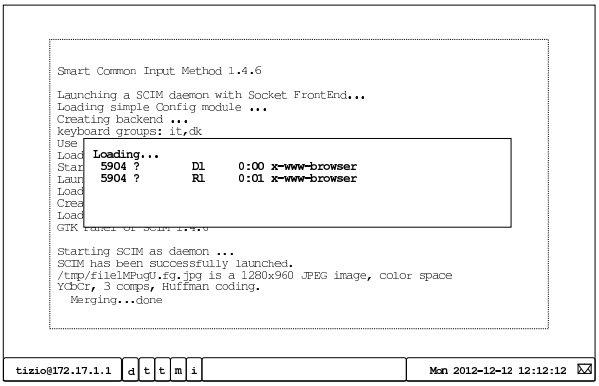


I messaggi che appaiono si possono scorrere all'indietro (anche con l'uso della rotellina del mouse, se disponibile), la finestra dei messaggi può essere eliminata ed eventualmente può essere riavviata successivamente.

Avvio dei programmi

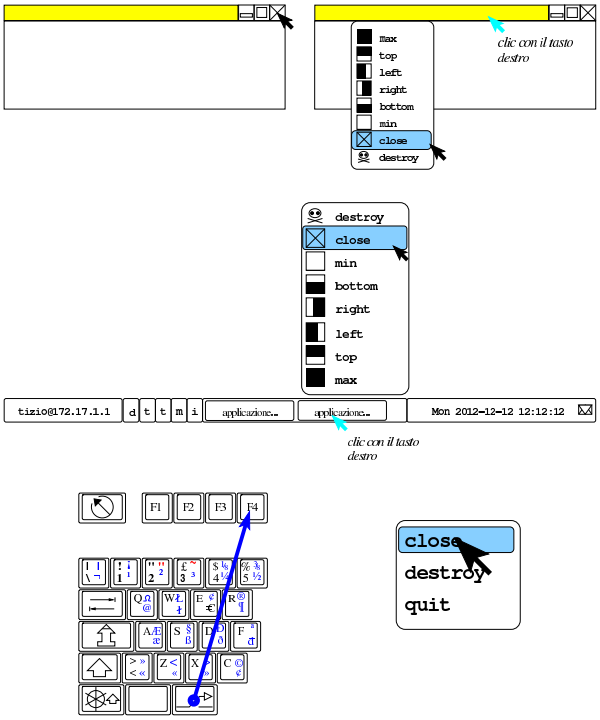
Il menù previsto per NLNX è organizzato in modo tale da facilitare l'avvio di alcune funzioni e programmi privilegiati. Nella maggior parte dei casi, nel menù sono previsti degli accorgimenti, come per esempio quello di mostrare un terminale con l'elenco dei processi più attivi, al momento dell'avvio delle applicazioni che ci mettono un po' di tempo prima di mostrarsi all'utente.

Figura u14.32. Attesa durante l'avvio di un programma.

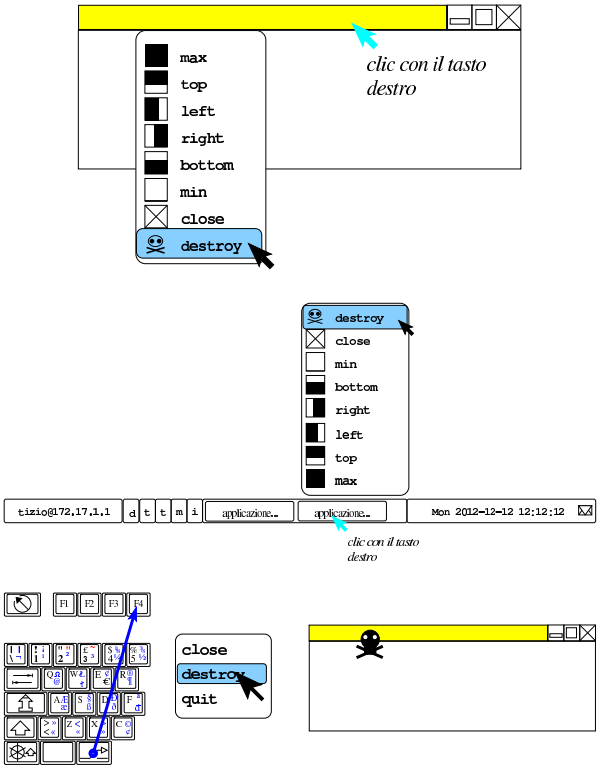


Chiusura dei programmi

Per concludere il funzionamento di un programma, oltre alle modalità previste dal programma stesso, è possibile fare un clic sul simbolo ☒ che appare sulla barra superiore della finestra, ma sono possibili anche altre opzioni, come sintetizzato dalle immagini successive:



Se si vuole eliminare un processo elaborativo, associato a un programma che si mostra graficamente, ma che non si chiude con metodi più delicati, si possono usare le maniere forti. Per esempio, attraverso questo metodo è possibile eliminare anche la finestra trasparente con i messaggi emessi dai programmi:



Configurazione del mouse

La configurazione attraverso '**`nlnxrc x config`**' non prevede l'indicazione del tipo di mouse, perché si utilizzano le informazioni trasmesse dal demone '**`gpm`**'; pertanto, la configurazione corretta di questo demone garantisce anche il funzionamento con X.

L'utente comune non ha la possibilità di configurare definitivamente la velocità o l'accelerazione del puntatore del mouse; per queste e altre cose simili deve usare ogni volta il comando '**`xset`**', per il quale appare qualche esempio nel menù di NLNX.

Sfondo

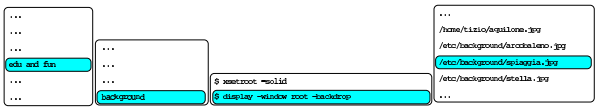
Lo script '**`xinitrc`**', all'avvio del sistema grafico fa una serie di operazioni, tra le quali c'è anche il caricamento dello sfondo, se configurato. Questa immagine è costituita dal file '`~/.wallpaper`' che eventualmente può contenere semplicemente un colore uniforme. Se questo file manca, viene utilizzato comunque un colore predefinito per lo sfondo.

Figura u14.39. Configurazione dello sfondo con colore uniforme, attraverso le voci del menù di Fvwm.


```
$ xsetroot -solid turquoise
```

Per definire il colore o l'immagine dello sfondo si possono usare le voci del menù di Fvwm; in particolare, per la selezione dell'immagine, è prevista la scansione del contenuto della directory '`/etc/background/`' e della directory personale dell'utente, alla ricerca di file con alcune estensioni tipiche.

Figura u14.40. Configurazione dell'immagine da usare come sfondo, attraverso le voci del menù di Fvwm.



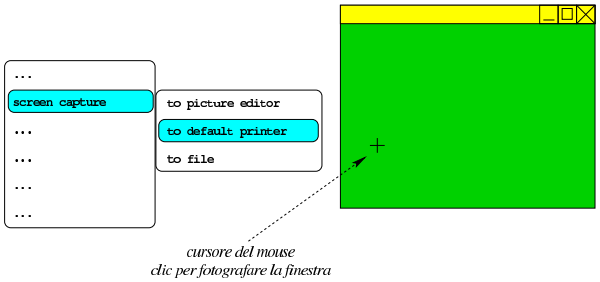

```
$ display -window root -backdrop /etc/background/spiaggia.jpg
```

Cattura dell'immagine dello schermo

Generalmente, con i programmi per il fotoritocco è possibile attivare una funzione per la cattura dell'immagine che appare sullo schermo o solo nell'ambito di una certa finestra. Tuttavia, questa operazione può essere laboriosa, mentre ci sono situazioni in cui poter salvare o stampare qualcosa che si vede attraverso il sistema grafico è importante. Per fare un esempio concreto, ci sono programmi che non dispongono di una funzione di stampa, come Gnuplot e in tal caso, la realizzazione di una foto dello schermo è l'unica possibilità di conservare qualcosa.

Nel menù di Fvwm appare una voce per la cattura delle immagini, che, a seconda dei casi, crea un file nella directory personale dell'utente, invia alla stampa il risultato, oppure crea un file temporaneo e avvia contestualmente un programma per il fotoritocco aprendo quel file. Una volta selezionata la funzione preferita, occorre puntare la finestra da catturare, quindi premere una volta il primo tasto del mouse.

Figura u14.41. Procedimento guidato dal menù di Fvwm per la cattura dell'immagine dello schermo.

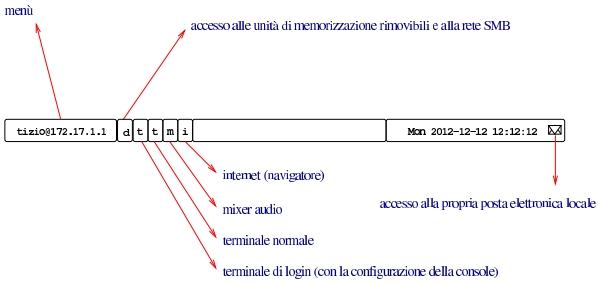


Quando si sceglie di salvare l'immagine in un file, si viene avvisati del nome utilizzato, che comunque corrisponde a un modello del tipo: '`screen.nn.jpg`'.

Bottoni della barra delle applicazioni

La barra delle applicazioni, che di solito appare nella parte inferiore della superficie grafica, contiene dei «bottoni» per accedere rapidamente a funzioni importanti. In particolare, sono presenti due bottoni con la lettera per avviare un terminale; la differenza sta nel fatto che il primo contiene una shell di *login* e come tale rilegge la configurazione come quando si accede al sistema.

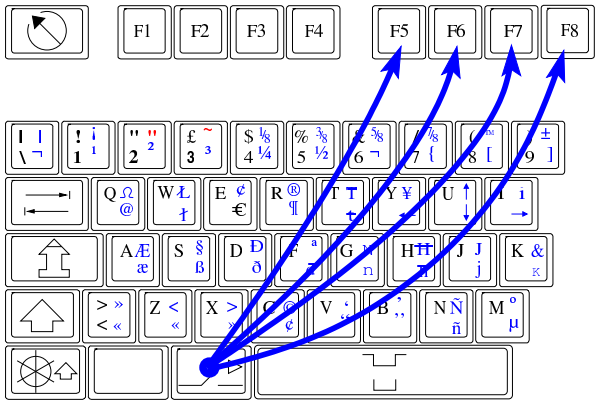
Figura u14.42. I «bottoni rapidi» incorporati nella barra delle applicazioni.



Registrazione dello schermo

Se è installato il programma Recordmydesktop e se è presente il file '`~/recordmydesktop.on`', attraverso alcune combinazioni di tasti è possibile controllare l'avvio, la sospensione e la conclusione di una registrazione audio-visuale, di tutto lo schermo grafico.

Figura u14.43. Controllo del funzionamento di Recordmydesktop.



Combinazione	Effetto
[*Alt F5*]	Inizia a registrare, ma senza audio.
[*Alt F6*]	Inizia a registrare normalmente, con l'audio.
[*Alt F7*]	Sospende o fa riprendere una registrazione in corso.
[*Alt F8*]	Conclude la registrazione.

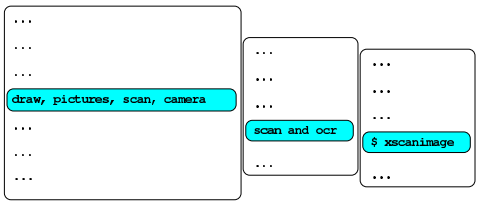
Il file '`~/recordmydesktop.on`' dovrebbe essere un immagine, la quale viene visualizzata al momento della conclusione, per un breve istante. Questa immagine potrebbe contenere delle note sul video, come per esempio il copyright.

La registrazione produce file denominati '`~/recordmydesktop[-nn].ogv`'. Tuttavia, alla conclusione della registrazione, occorre dare un po' di tempo a Recordmydesktop per produrre il file OGV finale.

Uso dello scanner

Se si dispone di uno scanner che può essere gestito da NLNX, per attivarne il controllo è sufficiente selezionare la voce *draw, pictures, scan, camera, scan and ocr, $ xscanimage.*

Figura u14.45. Avvio del programma Xscanimage, per la gestione dello scanner.



Avvio manuale del sistema grafico

L'accesso alla sessione grafica avviene di norma attraverso Xdm, anche quando si usa il sistema da DVD *live* o da altra unità in sola lettura. Tuttavia, quando si usa il sistema da DVD, la configurazione grafica iniziale è quella che viene determinata automaticamente (salva comunque la possibilità di intervenire con l'opzione di avvio del kernel '**`x_org_conf`**', come descritto nella sezione u0.1). Se così non dovesse funzionare, o se comunque lo si preferisce, si può far terminare il funzionamento di Xdm e utilizzare poi '**`startx`**', che in quel caso configura al volo la grafica.

```
# /etc/init.d/xdm stop [Invio]
```

...

```
$ startx [Invio]
```

Prima di far partire effettivamente la grafica, lo script '**`startx`**', avviato da un DVD *live* o da un altro file system in sola lettura, fa una serie di domande, come avviene durante la configurazione vera e propria che si farebbe con il comando '**`nlnxrc x config`**'.

Autorizzazione per l'accesso al servente X

Quando si utilizza Xdm, questo modifica ogni volta la chiave di autorizzazione associata al proprio protocollo nel file '`~/.Xauthority`' ('**`XDM-AUTHORIZATION-1`**'). Come conseguenza di ciò, se lo stesso utente ha più sessioni simultanee, su elaboratori differenti, condividendo la stessa directory personale, ovvero condividendo lo stesso file '`~/.Xauthority`', succede che solo l'ultima sessione aperta può risultare operativa in mondo corretto, mentre tutte le altre perdono la possibilità di avviare programmi nuovi.

Se esiste questo problema, l'utente che voglia aprire queste sessioni parallele deve avviare la grafica con l'ausilio dello script '**`startx`**', da una console, tenendo presente che se è in funzione Xdm occorre indicare uno schermo alternativo:

```
$ startx -- :1 [Invio]
```

Altre annotazioni

La configurazione di X è contenuta nel file '`/etc/X11/xorg.conf`'; il file '`/etc/X11/xorg.conf.vesa`' contiene la stessa configurazione e serve come modello quando si aggiorna il file '`/etc/X11/xorg.conf`'.

Per far sì che venga avviato il gestore di finestre Fvwm è stato modificato il file ‘`/etc/X11/xinit/xinitrc`’ e per garantire che rimanga così, nella directory ‘`/etc/X11/xinit/`’ ne è disponibile una copia di scorta che viene ricopiata automaticamente per opera dello script ‘`/etc/init.d/nlnx.config`’.

Per quanto riguarda la configurazione del gestore di finestre Fvwm, anche questa è stata riscritta (il file ‘`/etc/X11/fvwm/system.fvwm2rc`’) e lo script ‘`/etc/init.d/nlnx.config`’ provvede a mantenerla come disposto per NLNX, attraverso una sua copia.

Se si gradisce questo tipo di impostazione, le modifiche per il menù di Fvwm vanno apportate precisamente nel file ‘`/etc/X11/fvwm/system.fvwm2rc.nlnx`’ e copiate nel file ‘`/etc/X11/fvwm/system.fvwm2rc`’.

A proposito del menù di Fvwm realizzato per NLNX, si può osservare che, nella maggior parte dei casi, le voci non sono descrittive, ma contengono semplicemente il comando che andrebbe usato da un terminale a caratteri per avviare il programma. Questo tipo di impostazione serve a ridurre il distacco tra l’utilizzo del sistema a riga di comando e un utilizzo esclusivamente visuale-intuitivo.

Per approfondire l’argomento si può consultare il capitolo 28 dedicato alla gestione grafica.

Accesso ad altri file system con NLNX

Innesto di altri file system 57
Accesso semplificato alle unità rimovibili 57
Accesso a unità non rimovibili 58
Limitazione all’accesso 58
Accesso a risorse di rete SMB 59

`fstab` 57

NLNX, durante il funzionamento in modalità grafica, dispone di un sistema di innesto automatico dei file system contenuti in unità di memorizzazione USB rimovibili; per i dischi ottici o per altri tipi di unità, c’è comunque qualche accorgimento che viene incontro agli utenti. Tutte le unità che si vanno a innestare, si inseriscono in una sottodirectory di ‘`/mnt/`’.

Innesto di altri file system

Nel file ‘`/etc/fstab`’ di NLNX, alcune voci sono gestite in modo automatico, nel senso che tale file viene aggiornato dinamicamente, con l’inserimento o l’estrazione di unità esterne. Per tali unità, nella directory ‘`/mnt/`’ si creano e si eliminano automaticamente delle sottodirectory, in modo tale, per esempio, di innestare l’unità ‘`/dev/sdc2`’ nella directory ‘`/mnt/sdc2/`’.

Nel file ‘`/etc/fstab`’, queste voci gestite automaticamente hanno l’opzione ‘**`users`**’ (al plurale), con la quale si consente a ogni utente il distacco delle unità, anche se l’innesto non è stato eseguito dallo stesso:[1]

```
…
/dev/sdc1  /mnt/sda   auto  users,noauto  0  0
/dev/sdc2  /mnt/sda1  auto  users,noauto  0  0
…
```

Come accennato, la directory ‘`/mnt/`’ è inizialmente vuota, mentre uno script si occupa di inserire le sottodirectory necessarie e di togliere quelle che non servono più (oltre che aggiornare il file ‘`/etc/fstab`’ di conseguenza). Questo script viene pilotato automaticamente dal sistema uDev, che aggiorna al volo l’elenco dei file di dispositivo.

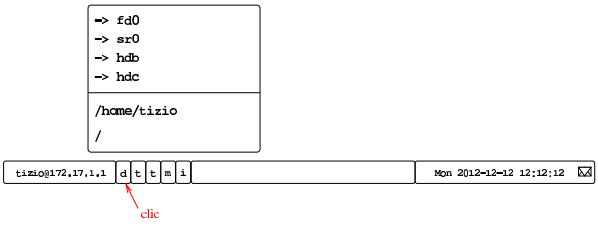
Le sottodirectory gestite automaticamente sono solo quelle che possono riferirsi a nomi di file di dispositivo di unità di memorizzazione, pertanto si possono comunque creare delle sottodirectory con nomi differenti per scopi propri, aggiornando anche il file ‘`/etc/fstab`’ manualmente. Per esempio si potrebbe voler aggiungere la voce seguente al file:

```
…
192.168.1.21:/  /mnt/192.168.1.21  nfs  users,noauto  0  0
…
```

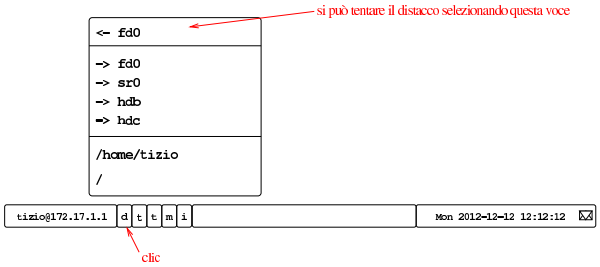
In tal caso andrebbe anche aggiunta la directory ‘`/mnt/192.168.1.21/`’, sapendo che lo script di aggiornamento di ‘`/mnt/`’ non la tocca, perché non corrisponde ai nomi dei file di dispositivo gestiti.

Accesso semplificato alle unità rimovibili

Per semplificare l’accesso alle unità rimovibili, in modo da non costringere gli utenti a usare i comandi ‘**`mount`**’ e ‘**`umount`**’, appare un pulsantino nella barra delle applicazioni, con la lettera [D].



Selezionando il pulsante grafico [D], si ottiene un menù con l'elenco delle unità rimovibili a cui si può accedere, assieme ad altre posizioni utili all'interno del file system. Eventualmente, se alcune unità risultano essere già innestate, si vedono le voci che consentirebbero di tentarne il distacco, come nell'esempio seguente dove si ipotizza che risulti già innestata un'unità nella directory '`/mnt/fd0/`':



Quando si seleziona una voce per accedere a una certa unità, se necessario viene innestata, quindi viene avviato un programma per poter intervenire nel suo contenuto. Di solito si tratta di Xfe, ma in mancanza di altro, può apparire anche solo una finestra di terminale.

Logicamente, si può ottenere il distacco di una unità solo quando questa non risulta più utilizzata. In particolare, quando l'innesto è avvenuto in modo automatico e a questo è seguito l'avvio di un programma come Xfe, alla conclusione del funzionamento di tali programmi, viene tentato un distacco automatico, che nel caso di unità servo-assistite, produce poi l'espulsione del supporto di memorizzazione.

Nella maggior parte dei casi, le unità rimovibili vengono innestate automaticamente, avviando contestualmente il programma Xfe, come descritto. Eventualmente, se dopo qualche secondo dopo l'inserimento dell'unità rimovibile non si vede apparire nulla, si può utilizzare la procedura già descritta.

Accesso a unità non rimovibili

Nell'elenco descritto nella sezione precedente, si trovano anche le unità non rimovibili, ma in tal caso, allo scopo di evitare errori, per ottenere l'innesto viene richiesto di inserire la parola d'ordine dell'amministratore. Naturalmente, l'utente comune che vuole accedere ugualmente a tali dati senza avere privilegi particolari, può farlo usando i comandi consueti, pertanto si vuole solo evitare che si innestino le partizioni di un disco locale senza essere consapevoli di ciò che si sta facendo.

Questa cautela deriva dalla possibilità che un disco locale contenga un file system NTFS, per il quale non c'è la massima compatibilità e si vuole evitare che un accesso non voluto esplicitamente possa comprometterne l'integrità.

Limitazione all'accesso

Per impedire a tutti gli utenti l'accesso a certe unità, non è possibile intervenire nel file '`/etc/fstab`', perché questo verrebbe aggiornato automaticamente, ripristinando i privilegi standard. Pertanto, è previsto un altro metodo, con cui si aggiungono dei file particolari nella directory '`/mnt/`'; file che logicamente un utente comune non possa cancellare. Si tratta di aggiungere un file vuoto con un nome corrispondente al modello seguente:

***nome_dispositivo*`.DISABLED`**

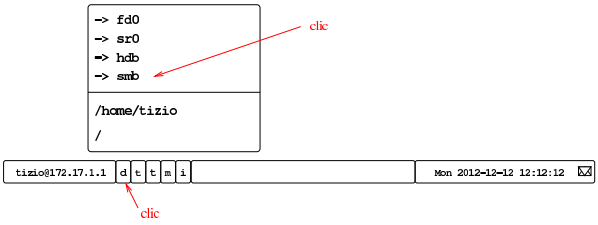
Per esempio, per impedire l'accesso alle unità a dischetti tradizionali, occorre creare i file '`/mnt/fd0.DISABLED`' e '`/mnt/fd1.DISABLED`'.

La presenza di questi file fa sì che non vengano create automaticamente le directory per l'innesto di tali dispositivi, impedendo in pratica agli utenti comuni di accedervi.

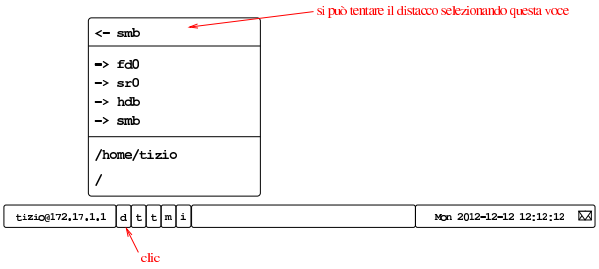
La gestione di questi file va fatta, evidentemente, attraverso degli script. Attualmente, se nella directory personale di un utente, esiste il file '`~/.nlnx/flags/normm`' (*no removable media*, nel momento dell'accesso attraverso la grafica, vengono creati al volo tutti i file che servono a impedire l'accesso a unità di memorizzazione esterne.[23]

Accesso a risorse di rete SMB

Se presso la propria rete locale (per la quale si deve essere instradati correttamente) sono presenti degli elaboratori MS-Windows configurati in modo da condividere alcune porzioni del proprio file system, secondo i protocolli SMB, dovrebbe essere possibile accedervi attraverso lo stesso menù previsto per le unità rimovibili:



Perché l'innesto possa avvenire, è necessaria la presenza della directory '`/mnt/smb/`', con i permessi 0777_8. Una volta innestato l'albero delle risorse individuate automaticamente, il distacco avviene come per le unità rimovibili:



[1] Originariamente si utilizzava l'opzione '**`user`**', al singolare, in modo che l'utente che innestava un file system fosse l'unico (a parte '**`root`**') che potesse anche eseguirne successivamente il distacco. Tuttavia, non sempre il sistema operativo si comporta conformemente e poteva succedere che tale operazione venisse autorizzata esclusivamente all'amministratore.

[2] Con una gestione opportuna dei permessi, si può fare in modo che gli utenti non possano cancellare o modificare il contenuto della directory '`~/.nlnx/flags/`'. In pratica, la configurazione predefinita di NLNX è tale per cui le directory personali degli utenti hanno i permessi 1771_8, pari a '**`rwxrwx--t`**', appartenendo all'utente '**`root`**' e al gruppo privato dell'utente rispettivo. In tal modo, gli utenti non hanno la facoltà di modificare i permessi della propria directory personale e non possono cancellare file o directory che non appartengono a loro stessi. Quindi, se le directory '`~/.nlnx/`' e '`~/.nlnx/flags/`' appartengono all'utente '**`root`**' e non ci sono i permessi di scrittura per il gruppo e gli altri utenti, nessuno oltre a '**`root`**' può intervenirvi.

[3] Purtroppo questo meccanismo non può essere applicato agli accessi da terminale testuale, in quanto gli script che possono essere messi in funzioni nella fase successiva all'identificazione dell'utente, operano soltanto con i privilegi dell'utente stesso; pertanto non ci sarebbe modo di creare i file '`/mnt/`*`x`*`.DISABLED`'.

nlnxrc 61

NLNX dispone di tre script principali, come descritto nella tabella successiva. Di questi, il più importante è '**nlnxrc**', del quale vengono poi riportate le tabelle con il riepilogo dei comandi di uso più comune. Si osservi che in quasi tutti i casi di utilizzo di '**nlnxrc**' occorre intervenire in qualità di utente '**root**'.

Tabella u16.1. Riepilogo degli script principali di NLNX.

Comando	Descrizione
`nlnx`	Visualizza la guida interna di NLNX in formato testo.
`appunti`	Scorre a video *a2* in formato testo.
`nlnxrc`	Menù per la configurazione e l'utilizzo delle funzionalità più importanti di NLNX.

Tabella u16.2. Script '**nlnxrc**': configurazione principale.

Comando	Descrizione
`nlnxrc network config`	Configurazione facilitata per l'accesso alla rete locale.
`nlnxrc mouse config`	Configurazione facilitata per l'uso del mouse.
`nlnxrc printer config`	Consente di ricreare il file '`/etc/printcap`' impostando la coda di stampa predefinita per la stampante che si usa in un certo momento.
`nlnxrc locale config`	Configurazione facilitata della localizzazione. La configurazione locale può essere distinta in base all'uso da console o da X.
`nlnxrc x config`	Crea un file '`/etc/X11/xorg.conf`', a partire da '`/etc/X11/xorg.conf.vesa`', consentendo di specificare il tipo di adattatore grafico.
`nlnxrc sound config`	Riconfigura automaticamente la gestione dell'audio attraverso ALSA.
`nlnxrc acpi spindown`	Configurazione facilitata del tempo di ritardo per lo spegnimento dei dischi, quando sono disponibili le funzionalità ACPI.
`nlnxrc date config`	Specifica a quale elaboratore rivolgersi per la sincronizzazione dell'orario del proprio.
`nlnxrc log-server config`	Specifica a quale elaboratore inviare copia del proprio registro del sistema.

Tabella u16.3. Script '**nlnxrc**': configurazione del servizio NIS.

Comando	Descrizione
`nlnxrc nis-server config` `nlnxrc nis-server unconf`	Attiva o disattiva il funzionamento in qualità di servente NIS.
`nlnxrc nis-server-users edit`	Per motivi di sicurezza, indica quali utenti comuni possono accedere direttamente all'elaboratore che offre il servizio NIS.
`nlnxrc nis stop`	Disattiva le funzioni NIS (sia come servente, sia come cliente).

Tabella u16.4. Script '**nlnxrc**': configurazione del servizio DHCP.

Comando	Descrizione
`nlnxrc dhcp-server config` `nlnxrc dhcp-server unconf`	Attiva o disattiva il funzionamento in qualità di servente DHCP.

Comando	Descrizione
`nlnxrc dhcp-server edit`	Modifica la configurazione del servente DHCP, intervenendo nel file di configurazione in modo libero.

Tabella u16.5. Script '**`nlnxrc`**': configurazione del servizio proxy HTTP.

Comando	Descrizione
`nlnxrc proxy clients` [*gruppo*]	Definisce l'elenco dei nodi di rete previsti per l'accesso al proprio servizio proxy HTTP.
`nlnxrc proxy access` [*gruppo*]	Seleziona i nodi di rete, tra quelli previsti, che possono accedere effettivamente al servizio.

Tabella u16.6. Script '**`nlnxrc`**': configurazione della stampa.

Comando	Descrizione
`nlnxrc printer config`	Consente di ricreare il file '`/etc/printcap`' impostando la coda di stampa predefinita per la stampante che si usa in un certo momento.
`nlnxrc print maxpages`	Consente di definire una quantità massima di pagine che possono essere stampate simultaneamente.
`nlnxrc printer clients` `nlnxrc printer access`	Nel primo caso consente di definire l'elenco dei nodi di rete previsti per l'accesso al proprio servizio di stampa; nel secondo consente di scegliere quali nodi, tra quelli previsti, possono accedere effettivamente al servizio di stampa.

Tabella u16.7. Script '**`nlnxrc`**': gestione delle utenze.

Comando	Descrizione
`nlnxrc user add` [*utente* ↩ ↪ [*home_dir* ↩ ↪ [*descrizione* ↩ ↪ [*parola_d'ordine*]]]] `nlnxrc user del` [*utente*]	Aggiunge o elimina un'utenza, secondo la procedura completa prevista da NLNX.
`nlnxrc user passwd` [*utente* ↩ ↪ [*parola_d'ordine*]]	Cambia la parola d'ordine di un utente, secondo la procedura completa prevista da NLNX.
`nlnxrc user info` `nlnxrc home info`	Consente di avere informazioni sugli utenti comuni, partendo, rispettivamente, da un elenco in ordine alfabetico del nominativo utente, oppure della directory personale.
`nlnxrc quota set`	Se è attiva la gestione delle quote di utilizzo della memoria di massa, consente di stabilire il limite di spazio per le utenze.
`nlnxrc quota report`	Se è attiva la gestione delle quote di utilizzo della memoria di massa, visualizza la situazione di tutti gli utenti, ordinando l'elenco partendo da quelli che stanno utilizzando più spazio.
`nlnxrc nis-server-users edit`	Per motivi di sicurezza, indica quali utenti comuni possono accedere direttamente all'elaboratore che offre il servizio NIS.
`nlnxrc machine add` `nlnxrc machine del` [*nome*]	Aggiunge o elimina un'utenza speciale, associata a un elaboratore MS-Windows, per la gestione degli accessi da tale sistema operativo, attraverso Samba.

Comando	Descrizione
`nlnxrc admin add` [*nominativo* ↩ ↪ [*descrizione* ↩ ↪ [*parola_d'ordine*]]] `nlnxrc admin del` [*nominativo*]	Aggiunge o elimina un'utenza amministrativa. Per la precisione, si ottiene un amministratore con nome ***nominativo*** dal lato NLNX e uno con nome **`win.`*nominativo*** dal lato MS-Windows.
`nlnxrc admin passwd` ↩ ↪ [*nominativo* [*parola_d'ordine*]]	Cambia la parola d'ordine di un amministratore, secondo la procedura completa prevista da NLNX.

Tabella u16.8. Script '**`nlnxrc`**': installazione e aggiornamento di NLNX.

Comando	Descrizione
`nlnxrc nlnx install`	Installa NLNX in una partizione già preparata. Può essere usato anche quando NLNX è già in funzione in un disco normale e se ne vuole installare una copia ulteriore in un altro disco.
`nlnxrc kernel make`	Ricompila un kernel personalizzato, generando un pacchetto Debian per il kernel e i suoi moduli principali, e altri pacchetti Debian per i moduli aggiuntivi, che vengono messi nella directory personale dell'utente *root*. Per avviare la compilazione è necessario che la directory corrente sia quella da cui si articolano i sorgenti del kernel.
`nlnxrc initrd make`	Dopo l'installazione di un nuovo kernel personalizzato, rigenera i file '`nlnxrd.img`' e '`onlnxrd.img`', in modo che contengano i moduli corretti. Questi file vengono installati automaticamente nei posti previsti da NLNX, ma di norma vanno poi copiati manualmente nella partizione di avvio.
`nlnxrc nlnx make`	Crea una nuova edizione di NLNX, con la possibilità di scegliere se si vogliono produrre solo i file-immagine o se si vuole arrivare direttamente a un DVD finale.
`nlnxrc nlnx rescue`	Crea un nuovo CD per l'avvio di emergenza di NLNX.

Tabella u16.9. Script '**`nlnxrc`**': controllo di VNC.

Comando	Descrizione
`nlnxrc vncs`	Si avvia preferibilmente da una console per attivare un servente VNC, definendo una parola d'ordine.
`nlnxrc vncsc`	Attiva un servente VNC, definendo una parola d'ordine, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc vncss`	Si avvia preferibilmente da una console per attivare un servente VNC condivisibile, definendo una parola d'ordine.
`nlnxrc vncssc`	Attiva un servente VNC condivisibile, definendo una parola d'ordine, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc vncv` ***nodo*** `nlnxrc vncv-ssh` ***nodo***	Consente di visualizzare il servente VNC in funzione presso il nodo indicato. La seconda delle due forme di utilizzo, implica la creazione di un tunnel SSH per garantire un collegamento cifrato.

Comando	Descrizione
`nlnxrc vncc` ***nodo*** `nlnxrc vncc-ssh` ***nodo***	Consente di interagire con il servente VNC in funzione presso il nodo indicato. La seconda delle due forme di utilizzo, implica la creazione di un tunnel SSH per garantire un collegamento cifrato.
`nlnxrc sharedx`	Attiva un servente VNC condivisibile, utilizzando una parola d'ordine predefinita, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc viewremotex` ***nodo***	Consente di visualizzare il servente VNC in funzione presso il nodo indicato, utilizzando la parola d'ordine predefinita. In pratica si usa per collegarsi a un servente VNC avviato con il comando '**`nlnxrc sharedx`**'.

Tabella u16.10. Script '**`nlnxrc`**': funzionalità relative a servizi offerti tramite il servente HTTP.

Comando	Descrizione
`nlnxrc mrtg config` `nlnxrc mrtg unconf`	Configura o disabilita le statistiche di utilizzo delle interfacce di rete, attraverso MRTG.
`nlnxrc webalizer update`	Aggiorna le statistiche di accesso al servente HTTP, prodotte con Webalizer.

Tabella u16.11. Script '**`nlnxrc`**': funzionalità generiche e accessorie.

Comando	Descrizione
`nlnxrc disc-file` ***file_immagine***	Incide un CD o un DVD (contenente dati) a partire dal file-immagine specificato.
`nlnxrc disc-dir` [***directory***]	Procede alla creazione di una copia della directory indicata, o della directory corrente, in un disco ottico. Se non ci sono già, vengono aggiunti i file '`autorun.inf`', '`autorun.bat`' e '`autorun.htm`'.
`nlnxrc disc blank` `nlnxrc disc blankfast`	Reinizializza un CD o un DVD riscrivibile. La seconda modalità esegue una cancellazione minima.
`nlnxrc audio record`	Inizia a registrare in un file temporaneo, ciò che proviene dall'adattatore audio.
`nlnxrc ssh hostkey` `nlnxrc ssh userkey`	Crea delle chiavi SSH nuove per l'elaboratore, o per l'utente.

Il comportamento del sistema può essere controllato attraverso una serie di opzioni di avvio, riepilogate dalle tabelle successive.

Tabella u17.1. Opzioni di avvio aggiuntive, specifiche del disco RAM iniziale.

Opzione	Descrizione
`n_boot=menu`|`auto`|`rescue`|`net` `n_boot=`***dispositivo***	Avvia la funzione indicata o il sistema contenuto nel file system di '`/dev/`***dispositivo***'.
`n_setupdelay=`***n***	Fa in modo che sia fatta una pausa di ***n*** secondi dopo il caricamento dei moduli previsti, in modo che le unità fisiche che vengono individuate con un po' di ritardo siano pronte prima di proseguire.
`n_modules=`***modulo***[`:`***modulo***]…	Consente di indicare un elenco di moduli da caricare, per i quali il sistema automatico non provvede autonomamente. I nomi dei moduli sono separati con il carattere due punti ('`:`'), senza inserire spazi tra un nome e l'altro. Non è possibile attribuire delle opzioni ai moduli da caricare.
`n_earlyswap=`[`0`|`1`]	In condizioni normali, le partizioni dei dischi locali che sembrano destinate allo scambio della memoria, vengono attivate già nelle primissime fasi dell'avvio. Eventualmente, con l'opzione '**`n_earlyswap=0`**' si richiede di evitare tale automatismo. Nel caso ci fosse invece il file '`nlnx.swp`' in una partizione locale, l'attivazione dello scambio della memoria avviene in modo indipendente, se non esiste già altra memoria di scambio attiva.

Tabella u17.2. Altre opzioni di avvio aggiuntive, alcune delle quali sono prese in considerazione anche dal disco RAM iniziale.

Opzione	Descrizione
`n_u_root=0`	Se si tratta di un file system in sola lettura, disabilita l'utenza '**`root`**'.
`n_u_default=0`	Se si tratta di un file system in sola lettura, disabilita le utenze predefinite '**`tizio`**', '**`caio`**',... '**`calpurnio`**'.
`n_xorg_conf=0`	Fa sì che il file '`/etc/X11/xdm/Xservers`' venga modificato per non avviare la grafica in modo automatico.

Opzione	Descrizione
`n_xorg_conf=auto`	Cancella il file ‘`/etc/X11/xorg.conf`’ e fa sì che il file ‘`/etc/X11/xdm/Xservers`’ venga modificato per avviare la grafica in modo automatico. In pratica, si vuole che il servente grafico si configuri automaticamente, ammesso che ne sia in grado.
`n_xorg_conf=default`	Ripristina il file ‘`/etc/X11/xorg.conf`’ secondo la configurazione predefinita di NLNX e fa sì che il file ‘`/etc/X11/xdm/Xservers`’ venga modificato per avviare la grafica in modo automatico.
`n_xorg_conf=`↩ ↪[*drv*], [*h*], [*v*], [*bit*], [*ris*]	Produce un file di configurazione ‘`/etc/X11/xorg.conf`’ secondo i valori forniti, lasciando gli altri al loro stato predefinito secondo NLNX, inoltre fa sì che il file ‘`/etc/X11/xdm/Xservers`’ venga modificato per avviare la grafica in modo automatico.
`n_auto_dm=1`	Avvia il menù di figura u13.5 che consente l’accesso alle utenze predefinite, senza l’inserimento della parola d’ordine (la sigla «dm» sta per *display manager*).
`n_nic=`***interfaccia_di_rete***	Specifica l’interfaccia di rete da utilizzare.
`n_ipv4=`***indirizzi_ipv4***	Specifica l’indirizzo IPv4 da utilizzare nella rete locale.
`n_subnet_mask=`***maschera_di_rete***	Specifica la maschera di rete da usare per l’indirizzo IPv4.
`n_router=`***indirizzo_ipv4***	Specifica l’indirizzo di un router da utilizzare per accedere alla rete esterna.
`n_root_path=`***percorso***	Specifica il percorso di un file system di rete (per esempio ‘`172.21.254.254:/opt/nlnx`’).
`n_ntp_server=`***indirizzo_ipv4*** `n_time_server=`***indirizzo_ipv4***	Specifica l’indirizzo di un servente in grado di fornire l’ora esatta.
`n_domain_name_server=`***indirizzo_ipv4***	Specifica l’indirizzo di un servente DNS (per la risoluzione dei nomi a dominio).
`n_log_server=`***indirizzo_ipv4***	Specifica l’indirizzo di un elaboratore a cui inviare i messaggi del registro di sistema.

Opzione	Descrizione
`n_lpr_server=`***indirizzo_ipv4*** `n_smb_prn=`***percorso_smb***	Nel primo caso specifica l’indirizzo di una stampante di rete o di un elaboratore che offre questo tipo di servizio. Nel secondo caso specifica il percorso SMB (del tipo ‘***//nodo/stampante***’) di una stampante condivisa di MS-Windows (se il nome della stampante contiene spazi, questi vanno sostituiti, ognuno, con la sequenza ‘**`%20`**’).
`n_smb_prn_user=`***nominativo*** `n_smb_prn_passwd=`***parola_d’ordine***	Nel caso sia utilizzata l’opzione ‘**`n_smb_prn`**’ e se l’accesso alla stampa richiede una forma di autenticazione, consente di indicare un nominativo e una parola d’ordine.
`n_lpr_filter=`***nome_filtro***	Specifica il nome del filtro da usare per la stampa. Il nome in questione corrisponde al nome di un file contenuto nella directory ‘`/etc/magicfilter/`’, togliendo la terminazione ‘`-filter`’, oppure al nome di un file PPD contenuto a partire da ‘`/usr/share/ppd/`’, togliendo l’estensione ‘`.ppd`’ o ‘`.ppd.gz`’.
`n_max_pages=`***n_pagine***	Specifica la quantità massima di pagine stampabili per volta.
`n_lpr_directory=`***directory***	Specifica il percorso assoluto di una directory, la quale deve avere i permessi di accesso 1777_8. Se si utilizza questa opzione e la directory esiste, le stampe «normali» vengono trasformate in file PDF collocati temporaneamente in questa directory.
`n_scanner=`***indirizzo_ipv4***	Specifica l’indirizzo IPv4 di uno scanner remoto.
`n_nis_domain=`***dominio_nis***	Specifica il dominio NIS.
`n_nis_server=`***indirizzo_ipv4***	Specifica l’indirizzo di un servente NIS.
`n_host_name=`***nome***	Specifica il nome dell’elaboratore.
`n_shutdown=`***ore*** : ***minuti*** [+***ore*** : ***minuti***]…	Specifica uno o più orari di spegnimento automatico.
`n_reboot=`***ore*** : ***minuti*** [+***ore*** : ***minuti***]…	Specifica uno o più orari di riavvio automatico.
`n_no_way_out=1`	Specifica che all’elaboratore cliente in questione non possa essere consentito raggiungere servizi di rete esterni, a parte quelli vitali, associati al servente NLNX.

Opzione	Descrizione
`n_ipv4_arp=`***ipv4*** `:` ***ethernet***	Fissa un indirizzo IPv4 di un elaboratore esterno a un indirizzo Ethernet, in modo che questa associazione non possa essere confusa. Va usata questa opzione esclusivamente per garantire che l'indirizzo IPv4 del servente rimanga associato all'indirizzo Ethernet corretto, quando nella rete locale c'è chi potrebbe associare per sbaglio lo stesso indirizzo IP a un elaboratore differente.
`n_ipwatch=0|1|2`	Se vi si associa il valore zero, disabilita la gestione, altrimenti attiva il demone '**`ipwatchd`**', allo scopo di individuare nella rete locale l'uso dello stesso indirizzo IPv4, da parte di altri nodi (questa funzione potrebbe non essere più disponibile).
`n_w_essid=`***nome***	Specifica l'identificativo ESSID di un'interfaccia di rete senza fili. Eventualmente si può indicare espressamente il nome '**`default`**' quando si vuole un collegamento automatico a una rete WiFi non cifrata.

`n_w_encryption=NONE`
`n_w_encryption=WEP`
`n_w_encryption=WPA-PSK`
	Specifica il sistema crittografico usato per la comunicazione attraverso un'interfaccia di rete senza fili.

`n_w_wep_key0=`***chiave*** `n_w_wep_key1=`***chiave*** `n_w_wep_key2=`***chiave*** `n_w_wep_key3=`***chiave***	Specifica le quattro chiavi del sistema crittografico WEP. La chiave può essere delimitata tra apici doppi per indicare che si tratta di una stringa, altrimenti si intende come una sequenza esadecimale.
`n_w_wpa_psk=`***chiave***	Specifica la chiave del sistema crittografico WPA-PSK. La chiave può essere delimitata tra apici doppi per indicare che si tratta di una stringa, altrimenti si intende come una sequenza esadecimale.

Preparazione delle partizioni

Piano di utilizzo delle partizioni 69
Sistema di emergenza: «rescue» 70
Suddivisione in partizioni 70
Inizializzazione delle partizioni 73

NLNX può essere installato in una partizione di un'unità di memorizzazione collegata al bus ATA interno o al bus USB.

Quando si installa NLNX in un'unità di memorizzazione comune, sia interna, sia esterna, per l'avvio si può usare SYSLINUX. Inoltre, come viene descritto nella sezione u28, per l'avvio dalla rete si usa PXELINUX.

Se si utilizza una partizione in una memoria allo stato solido, è necessario installare NLNX in modo che acceda al file system in sola lettura, per non bruciare rapidamente l'unità. Per lo stesso motivo, in tale memoria non va creata una partizione o un file per lo scambio della memoria virtuale.

Una volta installato in un disco che consenta l'accesso anche in scrittura, è possibile cambiare l'insieme dei pacchetti applicativi e riprodurre un nuovo DVD *live*, purché siano rimasti i programmi necessari per la registrazione su questo tipo di unità di memorizzazione.

Tabella u18.1. Cosa occorre usare per installare NLNX in un disco fisso comune o in un'unità esterna USB.

Comando	Descrizione
`fdisk` ***file_di_dispositivo***	Creazione o modifica delle partizioni.
`mkswap` ***file_di_dispositivo***	Inizializzazione di una partizione di scambio per la memoria virtuale.
`mkfs.ext3` ***file_di_dispositivo***	Inizializzazione di una partizione da usare per l'installazione del sistema operativo o di altri dati.
`mkfs.msdos` ***file_di_dispositivo***	Inizializzazione di una partizione da usare per l'avvio del sistema operativo, quando si utilizza SYSLINUX a questo proposito.
`nlnxrc nlnx install`	Comando di '**`nlnxrc`**' per eseguire la procedura di installazione.
`syslinux` `extlinux`	Predisposizione del sistema di avvio attraverso SYSLINUX.
`install-mbr`	Ricostruzione del settore di avvio generico.

Piano di utilizzo delle partizioni

Per l'installazione di NLNX viene proposta un'organizzazione particolare delle quattro partizioni primarie comuni, in modo da facilitare una sorta di standardizzazione, ammesso che non emergano esigenze specifiche particolari.

Tabella u18.2. Convenzione suggerita nella suddivisione in partizioni per l'installazione di NLNX.

Esempi di file di dispositivo	Utilizzo
'`/dev/hdx1`' '`/dev/sdx1`'	La prima partizione primaria, di tipo FAT16 (codice 6_{16}), contenente un file system Dos-FAT e utilizzata per collocare il sistema di avvio (kernel e SYSLINUX). Questa partizione deve avere una capacità di almeno 80 Mibyte, ma non può essere troppo grande; ed è assolutamente indispensabile la sua presenza se si deve usare SYSLINUX per l'avvio.

Esempi di file di dispositivo	Utilizzo
‘/dev/hdx2’ ‘/dev/sdx2’	La seconda partizione primaria di tipo Second-extended (codice 83_{16}), da usare per collocarvi il file system principale (preferibilmente di tipo Ext3).
‘/dev/hdx3’ ‘/dev/sdx3’	La terza partizione primaria da usare eventualmente per dati di vario tipo, oppure per un altro sistema operativo.
‘/dev/hdx4’ ‘/dev/sdx4’	La quarta partizione primaria da usare eventualmente per lo scambio della memoria virtuale (lo scambio della memoria virtuale non deve essere gestito su delle unità di memoria solida).

Per quanto riguarda la terza partizione primaria, se di questa non c’è bisogno subito, la si può omettere; tuttavia, nel caso si preveda la possibilità di averne bisogno in un momento successivo, si può predisporre inizialmente una quarta partizione per lo scambio della memoria virtuale, che in seguito potrebbe essere ridotta, per ricavare la terza partizione che inizialmente non serviva.

Sistema di emergenza: «rescue»

L’avvio di NLNX, in qualunque condizione, si avvale di un disco RAM iniziale, il quale contiene un sistema operativo minimo che può essere utilizzato autonomamente. Si ottiene l’avvio esclusivo di questo sistema minimo con la selezione della voce ‘**rescue**’.

Attraverso il sistema di emergenza è possibile svolgere molte delle operazioni che sono descritte in questo capitolo, contando su un utilizzo ridotto al minimo della memoria centrale. In particolare potrebbero essere create e inizializzate le partizioni, soprattutto quella per lo scambio della memoria virtuale. È possibile anche accedere a file system remoti attraverso il protocollo NFS ed è disponibile il programma ‘**partimage**’ per salvare e recuperare partizioni intere.

Suddivisione in partizioni

NLNX può essere installato in un disco, sia quando sta funzionando da DVD *live* (o da qualunque altro contesto in cui il file system principale è in sola lettura), sia quando è in funzione da un disco normale (in tal caso il file system è in lettura e scrittura). Ciò permette, per esempio, di installarlo da disco USB (lettura-scrittura) a disco ATA, da disco USB a un altro disco USB, o in altre combinazioni possibili.[1]

Prima di installare NLNX è necessario predisporre manualmente le partizioni nel disco che deve accoglierlo. Per questo è disponibile ‘**fdisk**’, con cui si deve definire una partizione per la memoria virtuale (tipo 82_{16}) e una per il file system (tipo 83_{16}). Eventualmente si può usare anche ‘**parted**’ per ridimensionare le partizioni già esistenti.

Quando si va a modificare la suddivisione in partizioni di un disco, occorre prima accertarsi di non utilizzarlo. L’errore più frequente che si commette sta nel dimenticare attiva una partizione per lo scambio della memoria virtuale. Ciò può succedere anche quando si avvia NLNX da un DVD *live*, perché se questo trova una partizione già predisposta per lo scambio della memoria virtuale, la utilizza. Pertanto, prima di intervenire in un disco con programmi come ‘**fdisk**’ e ‘**parted**’, occorre verificare di non utilizzare quel disco anche in tal modo. Si può verificare facilmente l’utilizzo di memoria di scambio con l’uso del comando ‘**free**’. A ogni modo, se ci si dimentica di questo o di altri accessi al disco, al termine delle modifiche, queste **non** sono prese in considerazione dal sistema, pertanto si può essere costretti a riavviare, o a ripeterle dopo che gli accessi sono stati esclusi. Eventualmente, per terminare l’uso di una memoria di scambio, basta il comando seguente:

```
# swapoff -a [Invio]
```

Viene mostrato un esempio sintetico di suddivisione in partizioni con ‘**fdisk**’, che si uniforma ai criteri descritti all’inizio del capitolo, riferito al primo disco SATA (‘/dev/sda’). Eventualmente si veda anche il capitolo 6 per una descrizione più dettagliata del procedimento.

```
# fdisk /dev/sda [Invio]

Command (m for help):
```

Il programma ‘**fdisk**’ accetta comandi composti da una sola lettera e per vederne un breve promemoria basta utilizzare il comando ‘**m**’.

```
Command (m for help): m [Invio]

  Command action
     a   toggle a bootable flag
     b   edit bsd disklabel
     c   toggle the dos compatibility flag
     d   delete a partition
     l   list known partition types
     m   print this menu
     n   add a new partition
     o   create a new empty DOS partition table
     p   print the partition table
     q   quit without saving changes
     s   create a new empty Sun disklabel
     t   change a partition's system id
     u   change display/entry units
     v   verify the partition table
     w   write table to disk and exit
     x   extra functionality (experts only)
```

La prima cosa che si fa normalmente è di visualizzare la situazione iniziale con il comando ‘**p**’:

```
Command (m for help): p [Invio]

  Disk /dev/sda: 164.5 GB, 164505600000 bytes
  255 heads, 63 sectors/track, 20000 cylinders
  Units = cylinders of 16065 * 512 = 8225280 bytes

     Device Boot   Start      End      Blocks   Id  System
  /dev/sda1   *        1    10000    80325000    b  W95 FAT32
  /dev/sda2        10001    20000    80325000    b  W95 FAT32
```

In questo caso, si preferisce cancellare le partizioni esistenti e ricominciare da zero:

```
Command (m for help): d [Invio]

Partition number (1-4): 1 [Invio]

Command (m for help): d [Invio]

  Selected partition 2

Command (m for help): p [Invio]
```

```
Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot   Start      End      Blocks   Id  System
```

A questo punto si definiscono la prima, la seconda e la quarta partizione, in base al piano della tabella u18.2, prospettando di non avere bisogno di una terza partizione per i dati:

```
Command (m for help): n[Invio]

Command action
   e   extended
   p   primary partition (1-4)
p[Invio]

Partition number (1-4): 1[Invio]

First cylinder (1-20000, default 1): 1[Invio]

Last cylinder or +size or +sizeM or +sizeK (1-20000, default
20000): +500M[Invio]

Command (m for help): p[Invio]

Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot   Start      End      Blocks   Id  System
/dev/sda1            1       60      481950   83  Linux
Command (m for help): n[Invio]

Command action
   e   extended
   p   primary partition (1-4)
p[Invio]

Partition number (1-4): 2[Invio]

First cylinder (61-20000, default 61): 61[Invio]

Last cylinder or +size or +sizeM or +sizeK (61-20000, default
20000): +150G[Invio]

Command (m for help): p[Invio]

Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot    Start         End      Blocks   Id  System
/dev/sda1             1          60      481950   83  Linux
/dev/sda2            61       19000   152135550   83  Linux
Command (m for help): n[Invio]

Command action
   e   extended
   p   primary partition (1-4)
p[Invio]

Partition number (1-4): 4[Invio]

First cylinder (19001-20000, default 19001): 19001[Invio]

Last cylinder or +size or +sizeM or +sizeK (19001-20000,
default 20000): 20000[Invio]

Command (m for help): p[Invio]

Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot    Start         End      Blocks   Id  System
/dev/sda1             1          60      481950   83  Linux
/dev/sda2            61       19000   152135550   83  Linux
/dev/sda3         19001       20000     8032500   83  Linux
```


A questo punto si deve modificare il tipo di partizione per '`/dev/sda1`' e '`/dev/sda4`', inoltre si deve rendere avviabile la prima:

```
Command (m for help): t[Invio]

Partition number (1-4): 1[Invio]

Hex code (type L to list codes): L

 0  Empty           1c  Hidden Win95 FA 70  DiskSecure Mult bb  Boot Wizard hid
 1  FAT12           1e  Hidden Win95 FA 75  PC/IX           be  Solaris boot
 2  XENIX root      24  NEC DOS         80  Old Minix       c1  DRDOS/sec (FAT-
 3  XENIX usr       39  Plan 9          81  Minix / old Lin c4  DRDOS/sec (FAT-
 4  FAT16 <32M      3c  PartitionMagic  82  Linux swap      c6  DRDOS/sec (FAT-
 5  Extended        40  Venix 80286     83  Linux           c7  Syrinx
 6  FAT16           41  PPC PReP Boot   84  OS/2 hidden C:  da  Non-FS data
 7  HPFS/NTFS       42  SFS             85  Linux extended  db  CP/M / CTOS / .
 8  AIX             4d  QNX4.x          86  NTFS volume set de  Dell Utility
 9  AIX bootable    4e  QNX4.x 2nd part 87  NTFS volume set df  BootIt
 a  OS/2 Boot Manag 4f  QNX4.x 3rd part 8e  Linux LVM       e1  DOS access
 b  Win95 FAT32     50  OnTrack DM      93  Amoeba          e3  DOS R/O
 c  Win95 FAT32 (LB 51  OnTrack DM6 Aux 94  Amoeba BBT      e4  SpeedStor
 e  Win95 FAT16 (LB 52  CP/M            9f  BSD/OS          eb  BeOS fs
 f  Win95 Ext'd (LB 53  OnTrack DM6 Aux a0  IBM Thinkpad hi ee  EFI GPT
10  OPUS            54  OnTrackDM6      a5  FreeBSD         ef  EFI (FAT-12/16/
11  Hidden FAT12    55  EZ-Drive        a6  OpenBSD         f0  Linux/PA-RISC b
12  Compaq diagnost 56  Golden Bow      a7  NeXTSTEP        f1  SpeedStor
14  Hidden FAT16 <3 5c  Priam Edisk     a8  Darwin UFS      f4  SpeedStor
16  Hidden FAT16    61  SpeedStor       a9  NetBSD          f2  DOS secondary
17  Hidden HPFS/NTF 63  GNU HURD or Sys ab  Darwin boot     fd  Linux raid auto
18  AST SmartSleep  64  Novell Netware  b7  BSDI fs         fe  LANstep
1b  Hidden Win95 FA 65  Novell Netware  b8  BSDI swap       ff  BBT

Hex code (type L to list codes): 6[Invio]

  Changed system type of partition 1 to 6 (FAT16)
Command (m for help): t[Invio]

Partition number (1-4): 4[Invio]

Hex code (type L to list codes): 82[Invio]

  Changed system type of partition 4 to 82 (Linux swap)
Command (m for help): p[Invio]

Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot   Start      End      Blocks   Id  System
/dev/sda1            1       60      481950    6  FAT16
/dev/sda2           61    19000   152135550   83  Linux
/dev/sda3        19001    20000     8032500   82  Linux swap
Command (m for help): a[Invio]

Partition number (1-4): 1[Invio]

Command (m for help): p[Invio]

Disk /dev/sda: 164.5 GB, 164505600000 bytes
255 heads, 63 sectors/track, 20000 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot   Start      End      Blocks   Id  System
/dev/sda1   *        1       60      481950    6  FAT16
/dev/sda2           61    19000   152135550   83  Linux
/dev/sda3        19001    20000     8032500   82  Linux swap
```

Per memorizzare le variazioni si conclude con il comando '**w**'; se invece si preferisce rinunciare, basta utilizzare il comando '**q**' che si limita a concludere l'esecuzione del programma annullando le operazioni svolte.

```
Command (m for help): w[Invio]

  The partition table has been altered!
  …
  Syncing disks.
  …
```

Inizializzazione delle partizioni

Dopo la suddivisione in partizioni, occorre inizializzare ciò che serve. Proseguendo secondo l'esempio di suddivisione appena proposto, si può intervenire così:

```
# mkfs.msdos /dev/sda1[Invio]
```

```
# mkfs.ext3 /dev/sda2 [Invio]

# mkswap /dev/sda4 [Invio]
```

[1] È il caso di precisare che non si può produrre un nuovo DVD *live* quando il sistema in funzione ha il file system principale in sola lettura, come quando sta già lavorando da un DVD *live* o da una memoria solida USB. Inoltre, l'installazione da un disco fisso normale (in lettura e scrittura) produce un risultato leggermente diverso, per ciò che riguarda il ripristino delle utenze.

Copia del sistema operativo

Copia di NLNX in una partizione di un disco «normale» 75
Copia in una directory 75
Configurazione 76
Configurazione alternativa 77
Scambio della memoria utilizzando un file 77

Dopo la suddivisione del disco in partizioni e dopo la loro inizializzazione, si può passare alla fase dell'installazione che copia il sistema operativo nel disco di destinazione. In generale questo procedimento è assistito da un comando apposito.

Copia di NLNX in una partizione di un disco «normale»

La copia di NLNX in una partizione di un disco fisso comune, viene eseguita con il comando seguente, indicando il file di dispositivo corrispondente alla partizione di destinazione, che deve essere stata preparata e inizializzata come già mostrato nel capitolo precedente:

```
# nlnxrc nlnx install [Invio]
```

Figura u19.1. Il comando '**`nlnxrc nlnx install`**' richiede di specificare il file di dispositivo corrispondente alla partizione nella quale installare il DVD.

```
.--------------------NLNX installation---------------------.
| Please insert the device file for the disk partition     |
| used as the destination or enter the already mounted     |
| mount-point.                                             |
| If you enter something like "/dev/..." it is assumed to  |
| be a device file; if you enter something like "/mnt/..." |
| it is assumed to be a mount-point, where you have        |
| already mounted your destination file system.            |
| .------------------------------------------------------. |
| |/dev/sd...                                            | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|               <  OK  >            <Cancel>               |
'----------------------------------------------------------'
```

Naturalmente, al posto di '**`/dev/sd...`**' occorre indicare il file di dispositivo che corrisponde alla partizione in cui si vuole installare il sistema operativo, cancellando i puntini finali superflui. Si osservi che l'esempio proposto dal programma sembra suggerire l'installazione in un disco SATA o un disco USB esterno, ma si può specificare anche un disco PATA tradizionale, mettendo il nome appropriato ('`/dev/hd`…'), oppure anche un insieme di dischi RAID ('`/dev/md`***`n`***').

Alcuni lettori DVD, all'apparenza perfettamente funzionanti, potrebbero non essere in grado di leggere tutto il contenuto di un disco, anche senza la segnalazione di alcun errore. È importante tenere presente il problema quando si utilizza NLNX in questo modo, ma soprattutto quando dal DVD si cerca di installarlo, perché nel verificarsi di questa ipotesi, ciò che si ottiene potrebbe essere solo una copia parziale del contenuto originale.

Copia in una directory

Disponendo della preparazione necessaria per farlo, nel caso lo si preferisca, è possibile installare NLNX in una struttura più articolata, composta da più dischi o semplicemente da più partizioni innestate tra di loro. Per questo scopo, occorre innestare inizialmente la partizione che deve ospitare il file system principale a partire da una sottodirectory di '`/mnt/`' (secondo il criterio normale di NLNX); successivamente occorre creare le directory ulteriori, a partire dalle quali si intende articolare il file system di destinazione, quindi occorre innestare manualmente le altre partizioni e infine procedere con la copia.

A titolo di esempio, si suppone di disporre di un disco PATA e di averlo suddiviso nel modo seguente:

Partizione	Scopo
`/dev/hda1`	partizione di avvio, da innestare nella directory '`/boot/`';
`/dev/hda2`	partizione del file system principale;
`/dev/hda5`	partizione da innestare nella directory '`/usr/`';
`/dev/hda6`	partizione da innestare nella directory '`/home/`';
`/dev/hda7`	partizione per lo scambio della memoria virtuale (*swap*).

Dopo aver inizializzato in modo appropriato le varie partizioni, queste vanno innestate secondo lo schema previsto, ma per farlo occorre anche creare le directory necessarie:

```
# mount /mnt/hda2 [Invio]

# mkdir /mnt/hda2/boot [Invio]

# mkdir /mnt/hda2/usr [Invio]

# mkdir /mnt/hda2/home [Invio]

# mount -t auto /dev/hda1 /mnt/hda2/boot [Invio]

# mount -t auto /dev/hda5 /mnt/hda2/usr [Invio]

# mount -t auto /dev/hda6 /mnt/hda2/home [Invio]
```

A questo punto, si può procedere con il comando di installazione, ma invece di indicare un file di dispositivo come destinazione, si deve specificare la directory da cui il tutto si articola: '`/mnt/hda2`'.

```
# nlnxrc nlnx install [Invio]
```

```
.--------------------NLNX installation---------------------.
| Please insert the device file for the disk partition     |
| used as the destination or enter the already mounted     |
| mount-point.                                             |
| If you enter something like "/dev/..." it is assumed to  |
| be a device file; if you enter something like "/mnt/..." |
| it is assumed to be a mount-point, where you have        |
| already mounted your destination file system.            |
| .------------------------------------------------------. |
| |/dev/sd...                                            | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|               < OK >            <Cancel>                 |
'----------------------------------------------------------'
```

Si deve correggere e indicare '`/mnt/hda2`':

```
/mnt/hda2 OK
```

Il resto procede normalmente.

Configurazione

Al termine della copia all'interno di una partizione, oltre a predisporre il sistema di avvio, come viene descritto nel capitolo successivo, è necessario intervenire in alcune parti della configurazione; per la precisione è necessario verificare il file '`etc/fstab`', all'interno del quale conviene anche indicare la partizione contenente la memoria virtuale. Si tratta delle prime tre righe che, secondo l'esempio proposto nel capitolo precedente, devono risultare alla fine nel modo seguente:

```
/dev/sda4  none          swap  sw                                  0 0
/dev/sda2  /             auto  defaults,usrquota,errors=remount-ro  0 1
/dev/sda2  /RO-FS/RW-FS  auto  defaults,usrquota,errors=remount-ro  0 0
…
```

Si osservi che è necessaria l'indicazione di due punti di innesto associati allo stesso file di dispositivo: reinnestando la partizione anche nella directory '`/RO-FS/RW-FS/`' è possibile poi ricreare un nuovo DVD *live* di NLNX, probabilmente dopo aver modificato qualcosa nella copia realizzata nella partizione del disco fisso; inoltre è possibile gestire il controllo delle quote di utilizzo del disco.

Al termine della copia di NLNX, lo script elimina la parola d'ordine agli utenti comuni predefiniti, in modo da non consentirne l'accesso, lasciando funzionante solo l'utenza dell'amministratore ('**`root`**'). Se prima dell'installazione la parola d'ordine è stata modificata, questo cambiamento viene mantenuto anche nella copia che viene installata. Se la parola d'ordine dell'utente '**`root`**' non è stata cambiata prima dell'installazione, alla prima occasione è necessario farlo, associandone eventualmente una agli utenti comuni che si vogliono usare. Si tenga conto anche del fatto che il sistema operativo che si ottiene installando NLNX prevede l'avvio automatico del servente OpenSSH, pertanto chiunque potrebbe accedere all'utenza '**`root`**' se la parola d'ordine originale non viene sostituita.

Configurazione alternativa

Benché l'esempio di configurazione del file '`/etc/fstab`', come mostrato nella sezione precedente, sia corretto, succede che se si installa NLNX in un disco esterno USB, non si possa sapere qual è esattamente il file di dispositivo associato a tale unità. Per esempio, se l'elaboratore in cui si innesta tale disco contiene internamente solo dischi PATA, il disco esterno dovrebbe risultare essere rappresentato da '`/dev/sda`', ma se al contrario quell'elaboratore ha già internamente un disco SATA, il disco esterno potrebbe essere '`/dev/sdb`'. Per poter definire una configurazione di compromesso nel file '`/etc/fstab`', si potrebbe sostituire l'esempio già visto con il contenuto seguente:

```
# FILE_SYSTEM  MOUNT_POINT  TYPE  OPTIONS  DUMP  PASS
/dev/sda4      none         swap  sw       0     0
/dev/sdb4      none         swap  sw       0     0
/dev/sdc4      none         swap  sw       0     0

rootfs  /             auto  defaults,usrquota,errors=remount-ro  0  1
rootfs  /RO-FS/RW-FS  auto  defaults,usrquota,errors=remount-ro  0  0
…
```

Come si può osservare, non potendo sapere dove si trova la partizione per lo scambio della memoria virtuale, se ne indicano diverse, contando sul fatto che quelle inesistenti non vengano utilizzate senza altre conseguenze; tuttavia, va sottolineato che con la soluzione proposta il file system principale non viene reinnestato nella directory '`/RO-FS/RW-FS`', perché il file di dispositivo '`rootfs`' non esiste.

Scambio della memoria utilizzando un file

In situazioni particolari, quando è necessario attivare lo scambio della memoria in un disco (*swap*) e non ci si può avvalere di una partizione, si può predisporre il file '`nlnx.swp`', della dimensione che si preferisce, collocato nella radice di un file system qualunque, purché disponibile. Eventualmente, benché sconsigliabile, in caso di estrema necessità questo file può essere collocato nella directory radice del file system usato per NLNX. Se si interviene così, non serve inizializzare il file, perché ciò viene fatto automaticamente nelle primissime fasi dell'avvio.

La sola presenza di un file con il nome '`nlnx.swp`' comporta l'attivazione automatica dello scambio della memoria su di esso, sovrascrivendo qualunque contenuto possegga già questo file. Dal momento che l'attivazione di questa funzione avviene nelle primissime fasi dell'avvio, se questo file si trova nello stesso file system di NLNX, diventa impossibile eseguire i controlli periodici di coerenza del file system, perché questo risulterebbe già in uso.

Installazione in un file-immagine

Installazione di un file-immagine in sola lettura 79
Installazione in sola lettura condivisa attraverso la rete 79
File-immagine in lettura e scrittura 80

È possibile installare NLNX in un file-immagine, invece che in una partizione, come si fa di solito con un sistema operativo. A questo proposito (con l'uso di un file-immagine) si presentano due alternative: l'installazione in sola lettura, oppure in lettura e scrittura.

Installazione di un file-immagine in sola lettura

È possibile utilizzare il file-immagine '`nlnx.img`' di NLNX per collocarlo in un file system dove potrebbe essere contenuto un altro sistema operativo (per esempio MS-Windows). Supponendo di disporre, nella partizione '`/dev/sda2`', di spazio sufficiente, si potrebbe semplicemente agire così, mentre si sta lavorando con un DVD *live*:

```
# mount /mnt/sda2 [Invio]

# cp /nlnx.img /mnt/sda2/nlnx.img [Invio]
```

È possibile integrare questa forma di installazione, aggiungendo il file per lo scambio della memoria virtuale, costituito da '`nlnx.swp`':

```
# gunzip < /nlnx.swp.gz > /mnt/sda2/nlnx.swp [Invio]
```

Se si intende usare il gruppo delle sole utenze predefinite, si può disporre anche del file '`nlnx.dat.gz`', da estrarre e collocare a fianco degli altri due, in modo tale da fornire delle directory personali a tali utenti, dove poter salvare i dati:

```
# gunzip < /nlnx.dat.gz > /mnt/sda2/nlnx.dat [Invio]
```

Al termine delle copie, il file system di destinazione va staccato:

```
# umount /mnt/sda2 [Invio]
```

Rimane però da organizzare un sistema di avvio.

Questo metodo di installazione può essere sfruttato anche se si può disporre in modo esclusivo di una partizione, o di tutta l'unità di memorizzazione di massa principale dell'elaboratore, perché consente un'installazione o un aggiornamento rapidi del sistema. Inoltre, diventa più semplice l'organizzazione dell'avvio, attraverso SYSLINUX, in una delle sue varianti (anche EXTLINUX, per esempio).

Il metodo descritto di installazione di NLNX, consentirebbe in teoria di utilizzare un file system di tipo Dos-FAT; tuttavia, considerato che si può usare al massimo la versione a 32 bit di questo file system, non è possibile collocare file più grandi di 4 Gibyte. Ecco perché, anche se si può disporre liberamente dell'unità di memorizzazione di massa principale, può essere conveniente o necessario avvalersi di un file system di tipo Second-extended (Ext2 o Ext3), organizzando l'avvio con EXTLINUX.

Installazione in sola lettura condivisa attraverso la rete

Esiste la possibilità di installare una copia di NLNX, in sola lettura, in modo tale che un gruppo di elaboratori senza disco fisso possa innestare il file system principale attraverso la rete.

Questo procedimento è spiegato nella sezione u28, ma il funzionamento che si ottiene è equivalente a quello in sola lettura descritto nelle sezioni precedenti.

File-immagine in lettura e scrittura

Un file-immagine da usare in lettura e scrittura è sostanzialmente la riproduzione di una partizione normale, solo che viene rappresentata da un file di grandi dimensioni. L'avvio di un file del genere avviene esattamente come se si trattasse di quello in sola lettura, pertanto il nome che deve avere il file nella destinazione è sempre '`nlnx.img`', con la sola differenza che, in questo caso, il file system che lo va a ospitare deve concedere l'accesso in scrittura. Per preparare un file di questo tipo occorre procedere manualmente, con i passi mostrati negli esempi successivi.

1. Per prima cosa occorre creare da qualche parte un file abbastanza grande, con l'aiuto di '**dd**':

   # **dd if=/dev/zero of=nlnx.img bs=1M count=20K** [*Invio*]

   In questo caso si crea il file '`nlnx.img`' da 20 Gibyte, nella directory corrente.

2. Il file creato va quindi inizializzato, come se fosse una partizione vera e propria:

   # **mkfs.ext3 -F nlnx.img** [*Invio*]

3. Il file-immagine, dopo l'inizializzazione, va innestato in modo da poter poi installare il sistema operativo:

   # **mkdir /mnt/loop** [*Invio*]

   # **mount -o loop -t auto nlnx.img /mnt/loop** [*Invio*]

4. Quindi si procede con l'installazione di NLNX, specificando che la si vuole all'interno di '`/mnt/loop`', e non di un file di dispositivo.

5. Al termine si deve distaccare il file-immagine e quindi occorre un modo per copiare il file stesso nel file system ospitante. Per esempio potrebbe essere compresso e poi estratto nella destinazione:

   # **umount /mnt/loop** [*Invio*]

   # **gzip -9 nlnx.img** [*Invio*]

Assieme al file-immagine con il file system di NLNX, è possibile aggiungere il file da usare per lo scambio della memoria virtuale, quello con il nome '`nlnx.swp`'. I passi successivi descrivono un esempio di creazione di tale file, della dimensione voluta.

1. # **dd if=/dev/zero of=nlnx.swp bs=1M count=1K** [*Invio*]

   In questo caso si crea il file '`nlnx.swp`' da 1 Gibyte, nella directory corrente.

2. Al termine occorre un modo per copiare il file stesso nel file system ospitante. Per esempio potrebbe essere compresso e poi estratto nella destinazione:

   # **gzip -9 nlnx.swp** [*Invio*]

Predisposizione del sistema di avvio

SYSLINUX 81
EXTLINUX 81
PXELINUX 82
Osservazioni sugli insiemi RAID 82
Osservazioni sulla convivenza di più sistemi operativi 82
Osservazioni sull'aggiornamento del kernel 83

Per l'avvio di NLNX, dopo la sua installazione, viene proposto SYSLINUX, assieme ai suoi derivati (EXTLINUX, PXELINUX). Quando l'avvio riguarda un'unità di memorizzazione locale (interna o esterna), i file del sistema di avvio vanno copiati nella radice di una partizione predisposta appositamente. Tale partizione può contenere un file system di tipo Dos-FAT o di tipo Ext2/Ext3 (in tal caso si usa per l'avvio SYSLINUX o EXTLINUX). I file necessari all'avvio sono quelli che si trovano nella directory '`/syslinux/`' di un disco ottico contenente NLNX. Se il sistema NLNX in funzione in quel momento è proprio nel disco ottico, la directory percepita è precisamente '`/initrd/r0/syslinux/`'.

SYSLINUX

SYSLINUX può essere usato per l'avvio di NLNX installato sia in dischi fissi PATA o SATA, sia in unità esterne, collegate attraverso il bus USB, utilizzando una partizione contenente un file system Dos-FAT. Pertanto, questa soluzione si presta solo per la creazione di una partizione di avvio, priva di altri dati (senza i file '`nlnx.*`'), perché non è possibile contenere file di dimensione maggiore ai 4 Gibyte.

Supponendo che la partizione in cui va installato il sistema di avvio sia '`/dev/sda1`', si comincia installando il codice necessario al funzionamento di SYSLINUX:

syslinux /dev/sda1 [*Invio*]

Si passa quindi all'innesto della partizione da usare per l'avvio:

mount /mnt/sda1 [*Invio*]

Si presume di avere già a disposizione la directory contenente i file necessari a SYSLINUX per l'avvio, inclusi i kernel e i dischi RAM iniziali; pertanto si passa alla loro copia:

cp -r ***directory_file_di_avvio*/* /mnt/sda1** [*Invio*]

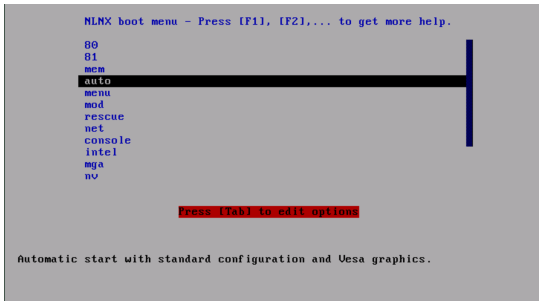
Si separa la partizione di avvio:

umount /mnt/sda1 [*Invio*]

Infine, occorre accertarsi di avere un settore di avvio adatto. Lo si ricrea facilmente con '**`install-mbr`**':[1]

install-mbr /dev/sda [*Invio*]

Quando poi si avvia il sistema, si ottiene una schermata simile a quella dell'avvio da disco ottico *live*:



Se successivamente si vuole intervenire nella configurazione dell'avvio, predisposto in questo modo con SYSLINUX, occorre modificare il file '`syslinux.cfg`'.

EXTLINUX

EXTLINUX è un'alternativa a SYSLINUX, da usare necessariamente se, nella stessa partizione di avvio, si vogliono collocare i file '`nlnx.*`', considerato che possono superare i 4 Gibyte di dimensione. Il tal caso, il file system contenuto nella partizione deve essere di tipo Ext2 o di tipo Ext3.

Supponendo che la partizione in cui va installato il sistema di avvio sia '`/dev/sda1`', si comincia installando il codice necessario al funzionamento di EXTLINUX, tenendo conto che a differenza di SYSLINUX, tale installazione richiede che la partizione di destinazione sia innestata:

```
# mount /mnt/sda1 [Invio]
```

```
# extlinux --install /mnt/sda1 [Invio]
```

Si presume di avere già a disposizione la directory contenente i file necessari a EXTLINUX per l'avvio (sono gli stessi file usati da SYSLINUX), inclusi i kernel e i dischi RAM iniziali; pertanto si passa alla loro copia:

```
# cp -r directory_file_di_avvio/* /mnt/sda1 [Invio]
```

Si separa la partizione di avvio:

```
# umount /mnt/sda1 [Invio]
```

Infine, occorre accertarsi di avere un settore di avvio adatto. Lo si ricrea facilmente con '**`install-mbr`**':[2]

```
# install-mbr /dev/sda [Invio]
```

Se successivamente si vuole intervenire nella configurazione dell'avvio, predisposto in questo modo con SYSLINUX, occorre modificare il file '`extlinux.conf`'.

PXELINUX

È possibile avviare NLNX attraverso la rete, con l'aiuto di PXELINUX. Questo procedimento è spiegato nella sezione u28.

Osservazioni sugli insiemi RAID

Il procedimento descritto in questi capitoli per l'installazione di NLNX, prevede l'uso di una partizione separata per gestire l'avvio del sistema operativo. Se si vuole installare NLNX in un insieme di dischi RAID (un file di dispositivo del tipo '`/dev/md`***n***'), ciò riguarda il file system principale, mentre **la partizione da usare per l'avvio deve rimanere estranea a tale gestione**.

Osservazioni sulla convivenza di più sistemi operativi

Quando si installa un sistema operativo è meglio essere prudenti: in generale è sconsigliabile tentare di far convivere due sistemi operativi diversi nello stesso disco o anche in dischi diversi ma collegati allo stesso bus ATA, a meno che **entrambi** siano predisposti per convivere pacificamente assieme. In generale, GNU/Linux funziona correttamente se vengono usati, alternativamente, anche altri sistemi sullo stesso elaboratore, ma possono essere gli altri sistemi operativi che non sono in grado di fare altrettanto. Piuttosto di rischiare, è sicuramente meglio installare NLNX in un disco esterno collegato con un bus USB.

Nel caso si debba installare NLNX in una partizione di un disco ATA, convivendo con un altro sistema operativo, il problema che si deve risolvere subito sta nel gestire l'avvio separato dei due sistemi, pertanto occorre conoscere la configurazione di GRUB 1 o di SYSLINUX per questo scopo (tenendo conto del rischio di non poter più avviare l'altro sistema operativo se si sbaglia qualcosa). Volendo evitare il problema conviene rinunciare del tutto all'installazione di GRUB 1 o di SYSLINUX, avviando NLNX con l'aiuto del DVD *live* o di un CD di emergenza, indicando la sigla '**`menu`**' all'avvio. La stessa soluzione (DVD *live* o CD di emergenza per l'avvio) varrebbe

installando NLNX in un disco USB esterno, in presenza di un BIOS che non è in grado di avviarlo.

Osservazioni sull'aggiornamento del kernel

Come descritto nella sezione dedicata alla modifica di NLNX (u23), se si vuole compilare un kernel personalizzato, occorre ricostruire la coppia di dischi RAM iniziali. Ciò che può sfuggire è il fatto che la copia materiale dei file del kernel e dei dischi RAM iniziali nella partizione usata per l'avvio, deve essere fatta manualmente.

[1] È indispensabile che alla partizione sia stato attivato l'indicatore di avvio, come già mostrato negli esempi all'inizio del capitolo.

[2] È indispensabile che alla partizione sia stato attivato l'indicatore di avvio, come già mostrato negli esempi all'inizio del capitolo.

Configurazione dell'avvio 85
Cenni alla configurazione successiva 86

`nlnxrd.img` 85 `onlnxrd.img` 85

NLNX si avvale di un disco RAM iniziale, costituito precisamente dai file '`nlnxrd.img`' e '`onlnxrd.img`' (il file usato dipende effettivamente dal kernel scelto per l'avvio) che si trovano ripetuti in vari punti, ma rimanendo sempre identici nelle varie copie. Il sistema contenuto all'interno del disco RAM iniziale serve principalmente per caricare i moduli necessari all'avvio, ma ne vengono caricati anche altri, in base alle caratteristiche dell'elaboratore individuate automaticamente.

Configurazione dell'avvio

Il sistema contenuto nel disco RAM iniziale e anche il sistema vero e proprio che con questo viene avviato, sono sensibili ad alcuni parametri del kernel che sono specifici di NLNX e si riconoscono perché hanno il prefisso iniziale '**`n_`**' (la lettera «n» sta logicamente per NLNX). In particolare, con il parametro '**`n_boot`**' è possibile istruire questo mini sistema su ciò che si intende fare. Per esempio, passando la voce '**`menu`**', si ottiene un menù di alternative; con la voce '**`rescue`**' si attiva la console per interagire con il sistema minimo; con la voce '**`net`**', si fa in modo che il mini sistema tenti di innestare un file system di rete, contenente un sistema NLNX in sola lettura. Se invece si passano voci come '**`sda`**', '**`sda1`**', '**`sda2`**',... si ottiene l'avvio automatico del sistema installato all'interno di '`/dev/sda`', '`/dev/sda1`', '`/dev/sda2`',...

La distribuzione include dei file di configurazione di esempio per SYSLINUX; tuttavia è il caso di osservare l'estratto seguente:

```
…
LABEL menu
 KERNEL vmlinuz
 APPEND n_boot=menu root=/dev/ram0 ro init=/linuxrc ↩
↪initrd=nlnxrd.img ramdisk_size=30720 n_setupdelay=8
…
LABEL hda1
 KERNEL vmlinuz
 APPEND n_boot=hda1 root=/dev/ram0 ro init=/linuxrc ↩
↪initrd=nlnxrd.img ramdisk_size=30720
…
```

La voce '**`menu`**' dei due esempi serve a richiedere al disco RAM l'attivazione di un menù, pertanto appare l'opzione '**`n_boot=menu`**'; tuttavia, si vede anche l'opzione '**`n_setupdelay=8`**', per ottenere un ritardo di otto secondi prima di costruire il menù stesso (serve per attendere il rilevamento di unità esterne USB eventuali). La voce '**`hda1`**' servirebbe per avviare un sistema NLNX installato nella prima partizione del primo disco PATA; pertanto, appare il parametro '**`n_boot=hda1`**', mentre in tal caso non è necessario alcun ritardo per l'individuazione del dispositivo relativo.

È importante osservare che se si utilizza il kernel più «vecchio», ovvero il file '`vmlinuz.old`', a questo va associato il file-immagine '`onlnxrd.img`', per il disco RAM iniziale. Per le opzioni si vedano le tabelle u17.1 e u17.2.

Seguono alcuni esempi di utilizzo di queste opzioni, dove si punta in particolare all'uso di '**`n_xorg_conf`**'. Si osservi che quando un'opzione può andare in conflitto con la configurazione automatica ottenuta tramite il DHCP, l'opzione specificata prevale.

- `n_boot=sdd7`
  avvia il sistema collocato nella partizione corrispondente al file di dispositivo '`/dev/sdd7`'.
- `n_modules=ehci_hcd:uhci_hcd`
  carica manualmente i moduli '**`ehci_hcd`**' e '**`uhci_hcd`**', per la gestione dei bus USB di tipo EHCI e UHCI. Il caricamento di tali moduli viene richiesto espressamente, perché evidentemente il

sistema automatico di riconoscimento dell'hardware non lo fa in modo corretto. Va osservato che se l'elaboratore offre entrambe le possibilità (EHCI e UHCI), i moduli vanno caricati in questo ordine.

- `n_xorg_conf=radeon,,,,`
  ridefinisce il file '`/etc/X11/xorg.conf`', partendo dalla configurazione standard di NLNX, stabilendo però che si tratta di un'interfaccia grafica di tipo «radeon».

- `n_xorg_conf=,30-90,50-100,,`
  ridefinisce il file '`/etc/X11/xorg.conf`', partendo dalla configurazione standard di NLNX, stabilendo però che la frequenza orizzontale può andare da 30 a 90 kHz, mentre la frequenza verticale può andare da 50 a 100 Hz.

- `n_xorg_conf=,,,32,`
  ridefinisce il file '`/etc/X11/xorg.conf`', partendo dalla configurazione standard di NLNX, stabilendo però che la profondità di colori deve essere da 32 bit.

- `n_xorg_conf=,,,,1024x768`
  ridefinisce il file '`/etc/X11/xorg.conf`', partendo dalla configurazione standard di NLNX, stabilendo però che la risoluzione grafica deve essere da 1024×768 punti.

- `n_xorg_conf=vesa,30-80,50-80,16,1280x1024`
  ridefinisce il file '`/etc/X11/xorg.conf`', stabilendo tutti i valori gestibili in questa fase.

Cenni alla configurazione successiva

Quando si installa NLNX in un disco comune (interno o esterno), conviene creare delle utenze personalizzate. Gli utenti '**`tizio`**', '**`caio`**',... '**`martino`**' e '**`calpurnio`**', pur non potendo accedere per mancanza di una parola d'ordine, dispongono comunque della loro directory personale: in alcune situazioni ciò può essere utile; in altre può essere considerato un problema di sicurezza. In generale, se gli utenti comuni predefiniti non servono, è meglio eliminare del tutto le loro directory personali, che corrispondono a '`/home/tizio/`', '`/home/caio/`',... '`/home/calpurnio/`'.

In ogni caso, questa modifica non pregiudica la produzione di un nuovo DVD *live*, perché queste directory personali verrebbero ricreate in modo automatico.

Se l'elaboratore in cui si lavora è connesso stabilmente a una rete locale, si può utilizzare il comando '**`nlnxrc network config`**' per la configurazione di una situazione comune, oppure si può arrivare anche a modificare lo script '`/etc/init.d/nlnx.network`' (rischiando però di dover rinunciare agli automatismi di NLNX). Può anche essere utile modificare i file '`/etc/hostname`' e '`/etc/mailname`'. Per esempio, se il proprio elaboratore deve avere il nome a dominio *dinkel.brot.dg*, il file '`/etc/hostname`' deve contenere la stringa '**`dinkel`**', mentre nel file '`/etc/mailname`' serve il nome a dominio completo.

Una volta installato NLNX, può essere necessario intervenire nel file di configurazione '`/etc/X11/xorg.conf`', perché in tal caso viene a mancare la configurazione automatica. Per la configurazione manuale si può usare il comando '**`nlnxrc x config`**', con i privilegi dell'utente '**`root`**'.

Attraverso il comando '**`nlnxrc x config`**' si ricrea il file '`/etc/X11/xorg.conf`' a partire da '`/etc/X11/xorg.conf.vesa`', modificando il tipo di adattatore grafico, la mappa della tastiera, i dati relativi alla scansione e alla profondità di colori.

Nel caso si intenda utilizzare il sistema APT (sezione 7.7) per fare delle modifiche sui pacchetti installati o per procedere a un aggiornamento di questi, si ricordi di modificare il file '`/etc/apt/sources.list`', indicando valori appropriati al proprio contesto.

Realizzazione di una propria variante di NLNX

Modifica dei pacchetti applicativi installati 87
Creazione del file-immagine della versione *live* 87
Identificazione dell'edizione 90
Ricompilazione del kernel 91
Installazione di software non libero 92

NLNX esiste con lo scopo di poter essere adattato facilmente alle proprie esigenze, aggiungendo e togliendo pacchetti a seconda dei bisogni. È possibile fare questo da una copia di NLNX installata secondo la procedura normale su una partizione di un disco fisso.

Si osservi che la modifica eventuale del kernel, seguita dalla sua installazione, deve essere completata con l'aggiornamento delle due versioni del disco RAM iniziale.

Modifica dei pacchetti applicativi installati

In generale, può essere conveniente partire da un'edizione di NLNX abbastanza completa, usando il programma '**`orphaner`**' per disinstallare tutto ciò che non serve. Il programma '**`orphaner`**' va usato con l'opzione '**`-a`**', per visualizzare l'elenco di tutti i pacchetti la cui rimozione non comporta problemi di dipendenze. Naturalmente si devono lasciare i programmi che servono per riprodurre il DVD.

```
# orphaner -a [Invio]
```

```
.--------------------Orphaner (r487)-----------------------.
| Select packages for removal or cancel to quit:           |
| .-----------(+)----------------------------------------. |
| |      [ ] myspell-sl           main/text              | |
| |      [ ] nlnx-80              main/text              | |
| |      [ ] nlnx-ko              main/text              | |
| |      [ ] nlnx-ok              main/text              | |
| |      [ ] nmapfe               main/net               | |
| |      [ ] nrg2iso              main/otherosfs         | |
| |      [ ] ntfsprogs            main/otherosfs         | |
| `----------.(+)----------------------------------------' |
|----------------------------------------------------------|
|    <   OK   >   <Simulate>   < Cancel >   <  Help  >     |
`----------------------------------------------------------'
```

Riquadro u23.2. Pacchetti specifici di NLNX.

NLNX potrebbe includere dei pacchetti Debian realizzati appositamente, per ricordare le dipendenze principali dell'edizione standard. Si tratta dei pacchetti il cui nome inizia per '**`nlnx-`**'.
Quando si tenta di adattare NLNX, questi pacchetti possono essere di intralcio, pertanto si possono rimuovere, senza temere effetti collaterali, a parte il fatto di non avere più riferimenti su cosa è indispensabile e su cosa invece è di intralcio o crea conflitti.
Eventualmente, si possono modificare gli stessi pacchetti intervenendo nel contenuto delle directory '`/etc/nlnx/debian-packages/*/`', ricompilandoli attraverso lo script '`/etc/nlnx/debian-packages/make-nlnx-packages`'. I file dei pacchetti generati si ottengono nella directory personale dell'utente '**`root`**':

```
# cd /etc/nlnx/debian-packages/ [Invio]
# ./make-nlnx-packages [Invio]
# cd /root [Invio]
# dpkg -i nlnx-...i386.deb [Invio]
```

Creazione del file-immagine della versione *live*

Una volta fatte le modifiche che si desiderano, si può usare il comando '**`nlnxrc nlnx make`**', senza argomenti, il quale innesta il file system principale nella directory '`/RO-FS/RW-FS/`' ed esegue le altre operazioni necessarie, dopo aver ottenuto alcune informazioni indispensabili:

```
# nlnxrc nlnx make [Invio]
```

```
.--------------------------NLNX--------------------------.
| Please note that producing the new disc or image file   |
| requires a lot of virtual memory (so you need a lot of  |
| swap space) and a lot of free space inside "/tmp/".     |
|                                                         |
| Is it ok? Should I continue?                            |
|                                                         |
|---------------------------------------------------------|
|                < Yes >         < No  >                  |
'---------------------------------------------------------'
```

Come avverte il riquadro iniziale, prima di poter avviare la procedura di creazione del nuovo disco o file-immagine, occorre verificare di avere abbastanza memoria virtuale e spazio su disco.

Dopo l'avvertimento iniziale si passa all'indicazione dell'edizione:

```
.-------NLNX release---------.
| Please insert the NLNX     |
| release identification     |
| string:                    |
| .------------------------. |
| |2010.02.22              | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Viene quindi chiesto di specificare il percorso del file-immagine da creare. Quanto predefinito corrisponde a un file nella directory '`/root/`':

```
.--------NLNX image file-----------.
| Please insert the NLNX image     |
| file pathname:                   |
| .------------------------------. |
| |/root/nlnx-2010.02.22.img     | |
| '------------------------------' |
|----------------------------------|
|       <  OK  >    <Cancel>       |
'----------------------------------'
```

A questo punto vengono fatte delle richieste che riguardano la configurazione locale iniziale del sistema che si va a creare. Si comincia con la lingua per il funzionamento da console:

```
.---------Console language selection-----------.
| Please select your language for the console: |
| .------------------------------------------. |
| |   en_US  default or previous selection   | |
| |   aa_DJ                                  | |
| |   af_ZA                                  | |
| |   br_FR                                  | |
| |   bs_BA                                  | |
| |   ca_ES                                  | |
| |   cs_CZ                                  | |
| |   da_DK                                  | |
| |   de_AT                                  | |
| |   de_BE                                  | |
| |   de_CH                                  | |
| |   de_DE                                  | |
| '---v(+)-----------------------------------' |
|----------------------------------------------|
|           <  OK  >      <Cancel>             |
'----------------------------------------------'
```

Il carattere usato per la console:

```
.--------------------------Console font selection---------------------------.
| Please select your console font:                                          |
| .---------------------------------------------------------------------. |
| | LatArCyrHeb-16+.psf.gz  previous_value                              | |
| | LatArCyrHeb-08.psf.gz   /usr/share/consolefonts/LatArCyrHeb-08.psf.gz  | |
| | LatArCyrHeb-14.psf.gz   /usr/share/consolefonts/LatArCyrHeb-14.psf.gz  | |
| | LatArCyrHeb-16.psf.gz   /usr/share/consolefonts/LatArCyrHeb-16.psf.gz  | |
| | LatArCyrHeb-16+.psf.gz  /usr/share/consolefonts/LatArCyrHeb-16+.psf.gz | |
| | LatArCyrHeb-19.psf.gz   /usr/share/consolefonts/LatArCyrHeb-19.psf.gz  | |
| '---------------------------------------------------------------------' |
|---------------------------------------------------------------------------|
|                    <  OK  >          <Cancel>                           |
'---------------------------------------------------------------------------'
```

La mappa per la tastiera durante l'utilizzo della console:


```
.--------------------Console keyboard map selection-------------------.
| Please select your console keyboard map:                           |
| .----------------------------------------------------------------. |
| |it-xorg       previous_value                                    | |
| |azerty        /usr/share/keymaps/i386/azerty/azerty.kmap.gz     | |
| |be2-latin1    /usr/share/keymaps/i386/azerty/be2-latin1.kmap.gz | |
| |be-latin1     /usr/share/keymaps/i386/azerty/be-latin1.kmap.gz  | |
| |fr            /usr/share/keymaps/i386/azerty/fr.kmap.gz         | |
| |fr-latin0     /usr/share/keymaps/i386/azerty/fr-latin0.kmap.gz  | |
| |fr-latin1     /usr/share/keymaps/i386/azerty/fr-latin1.kmap.gz  | |
| |fr-latin9     /usr/share/keymaps/i386/azerty/fr-latin9.kmap.gz  | |
| |fr-pc         /usr/share/keymaps/i386/azerty/fr-pc.kmap.gz      | |
| |fr-x11        /usr/share/keymaps/i386/azerty/fr-x11.kmap.gz     | |
| |mac-usb-be    /usr/share/keymaps/i386/azerty/mac-usb-be.kmap.gz | |
| |mac-usb-fr    /usr/share/keymaps/i386/azerty/mac-usb-fr.kmap.gz | |
| |mac-usb-it    /usr/share/keymaps/i386/azerty/mac-usb-it.kmap.gz | |
| |wangbe        /usr/share/keymaps/i386/azerty/wangbe.kmap.gz     | |
| |ANSI-dvorak   /usr/share/keymaps/i386/dvorak/ANSI-dvorak.kmap.gz| |
| |dvorak-classic /usr/share/keymaps/i386/dvorak/dvorak-classic.kmap| |
| 'v(+)------------------------------------------------------------' |
|--------------------------------------------------------------------|
|                  <  OK  >           <Cancel>                       |
'--------------------------------------------------------------------'
```

La lingua per il funzionamento in modo grafico:

```
.------------X language selection---------------.
| Please select your language for the X         |
| graphical environment.                        |
| Please note that if there will be an          |
| environment variable LANG_FOR_X defined, it   |
| will override this selection.                 |
| .---^(+)-----------------------------------. |
| |   pa_IN.UTF-8           UTF-8             | |
| |   pl_PL.UTF-8           UTF-8             | |
| |   pt_BR.UTF-8           UTF-8             | |
| |   pt_PT.UTF-8           UTF-8             | |
| |   ro_RO.UTF-8           UTF-8             | |
| |   ru_RU.UTF-8           UTF-8             | |
| |   ru_UA.UTF-8           UTF-8             | |
| |   se_NO.UTF-8           UTF-8             | |
| |   sid_ET.UTF-8          UTF-8             | |
| |   sk_SK.UTF-8           UTF-8             | |
| '---v(+)-----------------------------------' |
|-----------------------------------------------|
|           <  OK  >      <Cancel>              |
'-----------------------------------------------'
```

Il fuso orario:

```
.------------------Time zone selection----------------------.
| Please select your time zone:                             |
| .-------------------------------------------------------. |
| |   Europe/Rome              previous_value             | |
| |   Africa/Abidjan           Africa/Abidjan             | |
| |   Africa/Accra             Africa/Accra               | |
| |   Africa/Addis_Ababa       Africa/Addis_Ababa         | |
| |   Africa/Algiers           Africa/Algiers             | |
| |   Africa/Asmera            Africa/Asmera              | |
| |   Africa/Bamako            Africa/Bamako              | |
| |   Africa/Bangui            Africa/Bangui              | |
| |   Africa/Banjul            Africa/Banjul              | |
| |   Africa/Bissau            Africa/Bissau              | |
| |   Africa/Blantyre          Africa/Blantyre            | |
| |   Africa/Brazzaville       Africa/Brazzaville         | |
| '---v(+)-----------------------------------------------' |
|-----------------------------------------------------------|
|               <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

Il numero massimo di pagine stampabili per volta:

```
.--Set max lpr/lp printable pages----.
| Please insert how many pages are   |
| allowed to be printed for any      |
| single print; if you enter zero or |
| you leave blank, there is no       |
| limit:                             |
| .--------------------------------. |
| |15                              | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >    <Cancel>         |
'------------------------------------'
```

Infine viene chiesto che tipo di parola d'ordine usare per l'utente '**`root`**' e il nome dell'elaboratore:

```
.-----------root password selection--------------.
| Please, select the root password to be used:   |
| .---------------------------------------------. |
| |   nlnx     the default NLNX password       | |
| |   current  copy the current password       | |
| |   x        disable root password at all!   | |
| '---------------------------------------------' |
|-------------------------------------------------|
|           <  OK  >      <Cancel>               |
'-------------------------------------------------'

.---------hostname-----------.
| Please insert the hostname |
| to be used:                |
| .------------------------. |
| |nlnx                    | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

I file '`/etc/passwd`' e '`/etc/group`' vengono copiati nell'immagine del disco eliminando prima le utenze con numero UID da 1000 a 29999 (si osservi che gli utenti '**`tizio`**' e gli altri previsti per NLNX hanno numeri UID inferiori); di conseguenza il file '`/etc/shadow`' viene ricreato, usando la parola d'ordine scelta per l'utente '**`root`**', mentre per tutte le altre utenze standard ('**`tizio`**', '**`caio`**',... '**`martino`**', '**`calpurnio`**', e anche '**`user`**', '**`admin0`**', '**`admin1`**',... '**`admin9`**') viene usata la parola d'ordine standard: «nlnx».

Come si comprende, le utenze amministrative standard sono rese operative e questo per consentire di avere anche un'utenza '**`root`**' priva di parola d'ordine e quindi inutilizzabile.

La possibilità di disabilitare del tutto le parole d'ordine di '**`root`**' riguarda soprattutto la didattica, quando si vuole produrre un sistema su DVD *live*, oppure su unità a memoria solida USB, oppure ancora avviato direttamente dalla rete, senza l'intenzione di dare a utenti poco esperti la disponibilità di un'arma letale.

Lo script usato per la masterizzazione, prima della preparazione dei dati si occupa di sistemare alcune cose, come la **cancellazione della directory temporanea** e la creazione di una directory '`etc/`' adatta. Successivamente, la preparazione dei dati richiede un certo tempo:

```
...
[/etc/script/nlnxrc] "RW-FS/usr/share/dpatch"
[/etc/script/nlnxrc] "RW-FS/usr/share/dpkg"
[/etc/script/nlnxrc] "RW-FS/usr/share/dvdauthor"
[/etc/script/nlnxrc] "RW-FS/usr/share/dwww"
[/etc/script/nlnxrc] "RW-FS/usr/share/e2fsprogs"
[/etc/script/nlnxrc] "RW-FS/usr/share/emacs"
[/etc/script/nlnxrc] "RW-FS/usr/share/emacs21"
[/etc/script/nlnxrc] "RW-FS/usr/share/emoticons"
[/etc/script/nlnxrc] "RW-FS/usr/share/enchant"
[/etc/script/nlnxrc] "RW-FS/usr/share/enscript"
[/etc/script/nlnxrc] "RW-FS/usr/share/espeak-data"
[/etc/script/nlnxrc] "RW-FS/usr/share/et"
[/etc/script/nlnxrc] "RW-FS/usr/share/evince"
[/etc/script/nlnxrc] "RW-FS/usr/share/festival"
[/etc/script/nlnxrc] "RW-FS/usr/share/ffmpeg"
[/etc/script/nlnxrc] "RW-FS/usr/share/file"
[/etc/script/nlnxrc] "RW-FS/usr/share/file-roller"
[/etc/script/nlnxrc] "RW-FS/usr/share/fonts"
[/etc/script/nlnxrc] "RW-FS/usr/share/foomatic"
[/etc/script/nlnxrc] "RW-FS/usr/share/freecad"
[/etc/script/nlnxrc] "RW-FS/usr/share/fvwm"
...
```

Identificazione dell'edizione

L'edizione standard di NLNX è identificata dalla data di realizzazione, o di pubblicazione:

```
2012.12.31
```


Eventualmente, la mancanza del giorno indica la prima edizione del mese, mentre la mancanza anche del mese, indica che si tratta della prima edizione di quell'anno.

Le derivazioni di NLNX è bene siano identificabili come tali, possibilmente utilizzando un nome completamente diverso; tuttavia, se non gli si cambia il nome, è importante che già dalla stringa di edizione si capisca che si tratta di una variante, usando una forma abbastanza differente, che, possibilmente, consenta di intendere la natura della modifica o il contesto per il quale la derivazione è stata realizzata.

Ricompilazione del kernel

Il kernel standard di NLNX è fatto per gestire le situazioni più comuni; tuttavia ci sono componenti che non sono incluse, nemmeno come moduli. Per questo motivo, di norma sono presenti i sorgenti che sono serviti per la preparazione dei kernel utilizzati.

I sorgenti si trovano precisamente a partire da '`/usr/src/`'. Prima di passare alla configurazione con '**`make menuconfig`**' (o altro comando simile), conviene copiare il file di configurazione del kernel standard, che si trova nella directory '`/boot/`'. Per esempio, ipotizzando un kernel 2.6.23.9, si tratta di eseguire i passaggi seguenti:

```
# cp /boot/config-2.6.23.9 /usr/src/linux-2.6.23.9/.config
```
[*Invio*]

```
# cd /usr/src/linux-2.6.23.9
```
[*Invio*]

```
# make menuconfig
```
[*Invio*]

Al termine, una volta salvata una nuova configurazione, si passa alla compilazione, con l'aiuto di '**`nlnxrc`**':

```
# nlnxrc kernel make
```
[*Invio*]

```
.-----kernel revision--------.
| Please insert the kernel   |
| revision (must contain at  |
| least a number):           |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Viene richiesto di inserire una stringa di identificazione della revisione del kernel che deve contenere obbligatoriamente almeno un numero. Si potrebbe inserire qualcosa di simile all'esempio seguente:

```
01mio
```
[*Invio*]

Si osservi che si possono usare solo numeri e lettere (prima le cifre numeriche e poi le lettere alfabetiche). Se la stringa fornita non è accettabile, si ottiene una segnalazione di errore e occorre ripetere il comando.

Al termine, se tutto funziona come previsto, si ottengono alcuni pacchetti Debian nella directory personale dell'utente '**`root`**'. Seguendo l'esempio, si ottiene precisamente il file '`/root/linux-image-2.6.23.9_nl01mio_i386.deb`' e si dovrebbero ottenere i file relativi ai moduli aggiuntivi, tra cui, per esempio, '`/root/ndiswrapper-modules-2.6.23.9_3.3-3+nl01mio_i386.deb`' (ma questi moduli non sono indispensabili). Per installare i vari pacchetti si usa normalmente il programma '**`dpkg`**':

```
# cd /root
```
[*Invio*]

```
# dpkg -i linux-image-2.6.23.9_nl01mio_i386.deb
```
[*Invio*]

```
# dpkg -i ndiswrapper-modules-2.6.23.9_1.51-1+nl01mio_i386.deb
```
[*Invio*]

```
# dpkg -i kqemu-modules-2.6.23.9_1.3.0+nl01mio_i386.deb
```
[*Invio*]

...

L'installazione del kernel comporta l'aggiornamento dei collegamenti simbolici previsti nella directory radice: '`vmlinuz`' e '`vmlinuz.old`'. Purtroppo non è detto che rispecchino le proprie intenzioni ed eventualmente vanno modificati a mano. Dopo questa verifica, si può procedere alla ricostruzione del disco RAM iniziale, nelle sue due versioni, costituite dai file: '`nlnxrd.img`' e '`onlnxrd.img`'.

```
# nlnxrc initrd make [Invio]
```

> È molto importante ricordare di ricostruire sempre le immagini dei dischi RAM iniziali con il comando '**`nlnxrc initrd make`**', immediatamente dopo l'installazione di un kernel, perché **diversamente diventa impossibile avviare il sistema!** Inoltre occorre procedere a fare una copia dei file aggiornati nella partizione di avvio: i due kernel, presumibilmente nei nomi abbreviati '`vmlinuz`' e '`vmlinuz.old`', e i file '`nlnxrd.img`' e '`onlnxrd.img`'.

Quando viene creato un nuovo DVD *live*, con il comando '**`nlnxrc nlnx make`**', viene utilizzato sempre il kernel che risulta attivo, secondo quanto riportato dal comando '**`uname -a`**'; eventualmente si può controllare:

```
# uname -a [Invio]

Linux nlnx 2.6.23.9 #1 PREEMPT Dec 13 10:38:52 CET 2007 ↩
↪i686 GNU/Linux
```

Installazione di software non libero

Esiste del software che, a vario titolo, non è libero, ma del quale si può avere bisogno. Per aggiungere questo software a una versione installata di NLNX, conviene cercare delle versioni realizzate appositamente in formato di pacchetti Debian, in modo da non complicarsi troppo le cose. Naturalmente, occorre ricordare che se si installa del software non libero, non è possibile produrre una derivazione di NLNX da ridistribuire.

Una volta raccolti i pacchetti Debian da aggiungere alla propria installazione, è sufficiente utilizzare '**`dpkg`**' con l'opzione '**`-i`**', come nell'esempio seguente, indicando l'elenco di tutti i file da installare:

```
# dpkg -i msttcorefonts_2.3_all.deb ↩
↪        cabextract_1.2-2_i386.deb [Invio]

(Lettura del database …)
Spacchetto …
Spacchetto …
Configuro …
Configuro …
```

L'esempio mostra specificatamente l'installazione dei pacchetti attraverso cui vengono poi prelevati e installati i caratteri tipografici usati da MS-Windows, i quali non sono liberi. Questi pacchetti, di per sé, sarebbero liberi, ma '**`msttcorefonts`**' installa automaticamente i caratteri che non sono altrettanto liberi. L'installazione di '**`msttcorefonts`**' (e di ciò da cui questo dipende) consente di utilizzare tipi di carattere come «Arial» e altri, che diventano indispensabili per visualizzare correttamente certi file di MS-Office, specialmente quando l'incolonnamento delle voci è stata fatta imprudentemente con l'uso di sequenze di spazi orizzontali.

Configurazione di NLNX per l'interazione con il sistema

Configurazione locale 93
Configurazione di mouse e tastiera 95
Configurazione di X 96

`.LANG` 93 `.LANG_FOR_X` 93 `.profile` 93 `gpm.conf` 95 `LANG` 93 `xorg.conf` 96 `$LANG` 93 `$LANG_FOR_X` 93

La configurazione locale di NLNX, nella sua edizione standard, prevede inizialmente una variabile di ambiente '**`LANG`**' con il valore '**`en_US.UTF-8`**' per il funzionamento da console e '**`it_IT.UTF-8`**' per il funzionamento in modalità grafica, una mappa per la tastiera italiana e il fuso orario italiano; naturalmente, delle versioni modificate di questo lavoro possono essere pubblicate con una configurazione iniziale differente.

Una caratteristica speciale di NLNX per quanto riguarda la configurazione locale consiste nel separare il valore della variabile '**`LANG`**' per l'uso da console, rispetto a quello da modalità grafica. Ciò consente, per esempio, di utilizzare proficuamente la console, con una configurazione che prevede l'uso di una lingua con alfabeto latino, mentre in modalità grafica si può lavorare con alfabeti esotici, sia fonetici, sia ideografici.

Per quanto riguarda la gestione di X, è prevista una configurazione automatica, quando il file system è in sola lettura, ma in mancanza d'altro è disponibile una configurazione iniziale generica VESA. La configurazione iniziale (automatica o predefinita) può essere modificata con l'aiuto di '**`nlnxrc`**', oltre che con l'intervento diretto sul file '`/etc/X11/xorg.conf`'.

Configurazione locale

È possibile modificare la configurazione locale con il comando '**`nlnxrc locale config`**', che cambia comportamento se avviato dall'utente '**`root`**' o da un utente comune:

```
# nlnxrc locale config [Invio]
```

La prima richiesta a cui si viene sottoposti riguarda la selezione della stringa da assegnare alla variabile '**`LANG`**', che riguarda direttamente il comportamento della console. Questo valore viene poi salvato nel file '`/etc/nlnx/LANG`': se si accede nuovamente al sistema (eventualmente in un'altra console virtuale), la variabile '**`LANG`**' viene impostata in base al contenuto di questo file (ciò avviene attraverso '`/etc/profile`').

Figura u24.1. Configurazione della variabile '**`LANG`**'.

```
.---------Console language selection-----------.
| Please select your language for the console: |
| .------------------------------------------. |
| |   en_US  default or previous selection   | |
| |   aa_DJ                                  | |
| |   af_ZA                                  | |
| |   br_FR                                  | |
| |   ca_ES                                  | |
| |   da_DK                                  | |
| |   de_AT                                  | |
| |   de_BE                                  | |
| |   de_CH                                  | |
| |   de_DE                                  | |
| '---.(+)-----------------------------------' |
|----------------------------------------------|
|          <  OK  >        <Cancel>           |
'----------------------------------------------'
```

Si osservi che vengono proposti solo i linguaggi che possono essere utilizzati effettivamente in una console di un sistema GNU/Linux, in quanto è visualizzabile solo un insieme ristretto di caratteri.

Dopo la variabile '**`LANG`**' si passa alla selezione dei caratteri usati per la rappresentazione sullo schermo della console. Viene proposto un elenco di alcuni insiemi validi in generale per i linguaggi che si

possono selezionare. La scelta si ripercuote immediatamente su tutte le console virtuali.

Figura u24.2. Configurazione dei caratteri usati per la console.

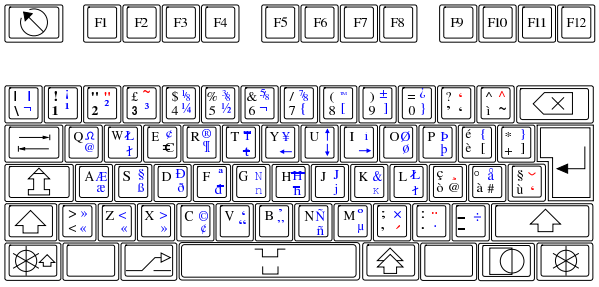
```
.--------------------------Console font selection--------------------------.
| Please select your console font:                                         |
| .----------------------------------------------------------------------. |
| | LatArCyrHeb-16+.psf.gz  previous_value                               | |
| | LatArCyrHeb-08.psf.gz   /usr/share/consolefonts/LatArCyrHeb-08.psf.gz  | |
| | LatArCyrHeb-14.psf.gz   /usr/share/consolefonts/LatArCyrHeb-14.psf.gz  | |
| | LatArCyrHeb-16.psf.gz   /usr/share/consolefonts/LatArCyrHeb-16.psf.gz  | |
| | LatArCyrHeb-16+.psf.gz  /usr/share/consolefonts/LatArCyrHeb-16+.psf.gz | |
| | LatArCyrHeb-19.psf.gz   /usr/share/consolefonts/LatArCyrHeb-19.psf.gz  | |
| `----------------------------------------------------------------------' |
|----------------------------------------------------------------------|
|                    <  OK  >           <Cancel>                       |
`----------------------------------------------------------------------'
```

Successivamente viene richiesto di specificare la mappa della tastiera, da usare per la console. Nel caso di una tastiera italiana, è consigliabile la scelta della voce '**`it-xorg`**'. Anche questa selezione si ripercuote immediatamente su tutte le console virtuali.

Figura u24.3. Configurazione della mappa della tastiera per la console.

```
.-----------------Console keyboard map selection-------------------.
| Please select your console keyboard map:                         |
| .---------------------------------------------------------------. |
| |it-xorg     previous_value                                     | |
| |azerty      /usr/share/keymaps/i386/azerty/azerty.kmap.gz      | |
| |be2-latin1  /usr/share/keymaps/i386/azerty/be2-latin1.kmap.gz  | |
| |be-latin1   /usr/share/keymaps/i386/azerty/be-latin1.kmap.gz   | |
| |fr          /usr/share/keymaps/i386/azerty/fr.kmap.gz          | |
| |fr-latin0   /usr/share/keymaps/i386/azerty/fr-latin0.kmap.gz   | |
| |fr-latin1   /usr/share/keymaps/i386/azerty/fr-latin1.kmap.gz   | |
| |fr-latin9   /usr/share/keymaps/i386/azerty/fr-latin9.kmap.gz   | |
| |fr-pc       /usr/share/keymaps/i386/azerty/fr-pc.kmap.gz       | |
| |fr-x11      /usr/share/keymaps/i386/azerty/fr-x11.kmap.gz      | |
| |mac-usb-be  /usr/share/keymaps/i386/azerty/mac-usb-be.kmap.gz  | |
| `.(+)-------------------------------------------------------------' |
|-------------------------------------------------------------------|
|                 <  OK  >            <Cancel>                      |
`-------------------------------------------------------------------'
```

Figura u24.4. NLNX si presenta inizialmente con una mappa della tastiera per la console che è stata realizzata appositamente, in modo da essere il più simile possibile a quella usata per la lingua italiana con la grafica di X (si veda eventualmente la sezione 14.5). Volendo selezionare espressamente questo tipo di mappa, occorre scegliere la voce '**`it-xorg`**'.



Il fuso orario viene indicato selezionando una città di riferimento.

Figura u24.5. Configurazione del fuso orario.

```
.-------------------Time zone selection---------------------.
| Please select your time zone:                             |
| .-------------------------------------------------------. |
| |   Europe/Rome              previous_value             | |
| |   Africa/Abidjan           Africa/Abidjan             | |
| |   Africa/Accra             Africa/Accra               | |
| |   Africa/Addis_Ababa       Africa/Addis_Ababa         | |
| |   Africa/Algiers           Africa/Algiers             | |
| |   Africa/Asmera            Africa/Asmera              | |
| |   Africa/Bamako            Africa/Bamako              | |
| |   Africa/Bangui            Africa/Bangui              | |
| |   Africa/Banjul            Africa/Banjul              | |
| |   Africa/Bissau            Africa/Bissau              | |
| |   Africa/Blantyre          Africa/Blantyre            | |
| |   Africa/Brazzaville       Africa/Brazzaville         | |
| `---v(+)----------------------------------------------' |
|-------------------------------------------------------------|
|              <  OK  >         <Cancel>                     |
`-------------------------------------------------------------'
```

Al termine viene offerto di specificare il valore da assegnare alla variabile di ambiente '**`LANG_FOR_X`**'; vengono proposti solo linguaggi che utilizzano la codifica UTF-8.

Figura u24.6. Configurazione della variabile '**`LANG_FOR_X`**' che si riflette nella configurazione locale del sistema grafico.

```
.------------X language selection----------------.
| Please select your language for the X         |
| graphical environment.                        |
| Please note that if there will be an          |
| environment variable LANG_FOR_X defined, it   |
| will override this selection.                 |
| .----(+)-------------------------------------. |
| |   pa_IN.UTF-8           UTF-8              | |
| |   pl_PL.UTF-8           UTF-8              | |
| |   pt_BR.UTF-8           UTF-8              | |
| |   pt_PT.UTF-8           UTF-8              | |
| |   ro_RO.UTF-8           UTF-8              | |
| |   ru_RU.UTF-8           UTF-8              | |
| |   ru_UA.UTF-8           UTF-8              | |
| |   se_NO.UTF-8           UTF-8              | |
| |   sk_SK.UTF-8           UTF-8              | |
| |   sl_SI.UTF-8           UTF-8              | |
| |   so_ET.UTF-8           UTF-8              | |
| `----.(+)-------------------------------------' |
|-------------------------------------------------|
|          <  OK  >       <Cancel>               |
`-------------------------------------------------'
```

Si osservi che gli utenti comuni possono utilizzare il comando '**`nlnxrc locale config`**', per definire il contenuto delle variabili '**`LANG`**' e '**`LANG_FOR_X`**', perché lo script '`~/.profile`' va a leggere il contenuto dei file '`~/.LANG`' e '`~/.LANG_FOR_X`', nei quali vengono annotate queste informazioni:

```
$ nlnxrc locale config [Invio]
```

L'utente comune non può cambiare i caratteri usati per la visualizzazione attraverso la console e non può nemmeno cambiare la mappa della tastiera, mentre rimane questa possibilità quando utilizza la grafica.

Configurazione di mouse e tastiera

La configurazione iniziale della tastiera è prevista secondo le convenzioni della distribuzione GNU/Linux Debian, nel file '`/etc/console/boottime.kmap.gz`'. La mappa predefinita di NLNX è diversa da quella standard per la lingua italiana usata dalla distribuzione Debian; pertanto, se si aggiorna il pacchetto relativo alla gestione della tastiera, può darsi che la mappa venga rimpiazzata con quella comune, perdendo le estensioni previste per NLNX. Se necessario, la mappa estesa si trova in '`/usr/share/keymaps/i386/qwerty/it-xorg.kmap.gz`', oppure si utilizza nuovamente '**`nlnxrc locale config`**'.

```
# cp /usr/share/keymaps/i386/qwerty/it-xorg.kmap.gz ↩
↪    /etc/console/boottime.kmap.gz [Invio]
```

La gestione del mouse è sottoposta al controllo di GPM e il sistema grafico, X, utilizza le informazioni generate da GPM stesso, senza accedere direttamente al mouse. La configurazione predefinita di NLNX prevede l'uso di un mouse di tipo PS/2 con rotellina (precisamente un mouse IntelliMouse PS/2), che è perfettamente compatibile con un mouse PS/2 normale, anche se fosse a due tasti, ma si può utilizzare il comando '**nlnxrc mouse config**' per attivare la gestione di un mouse differente:

```
# nlnxrc mouse config [Invio]
```

```
.------------------------Mouse selection--------------------------.
| Please select the mouse command:                                |
| .-------------------------------------------------------------. |
| |gpm -t imps2 -m /dev/psaux -R ms3  previous_value            | |
| |gpm -t imps2 -m /dev/psaux -R ms3  IM PS/2 (wheel)           | |
| |gpm -t ps2   -m /dev/psaux -R ms3  PS/2                      | |
| |gpm -t ms3   -m /dev/ttyS0 -R ms3  MS IM serial COM1: (wheel)| |
| |gpm -t ms3   -m /dev/ttyS1 -R ms3  MS IM serial COM2: (wheel)| |
| |gpm -t ms    -m /dev/ttyS0 -R ms3  MS serial COM1:           | |
| |gpm -t ms    -m /dev/ttyS1 -R ms3  MS serial COM2:           | |
| |gpm -t msc   -m /dev/ttyS0 -R ms3  MSystems serial COM1:     | |
| |gpm -t msc   -m /dev/ttyS1 -R ms3  MSystems serial COM2:     | |
| `-------------------------------------------------------------' |
|-----------------------------------------------------------------|
|                 <  OK  >           <Cancel>                     |
`-----------------------------------------------------------------'
```

Eventualmente si può intervenire manualmente nel file '`/etc/gpm.conf`', cambiando ciò che serve; se si agisce in questo modo, dopo la modifica, ovviamente, si deve riavviare il demone '**gpm**' attraverso lo script previsto dalla distribuzione:

```
# /etc/init.d/gpm stop [Invio]
```

```
# /etc/init.d/gpm start [Invio]
```

Se funziona il mouse su una console, funziona di conseguenza anche con X.

Può capitare che il demone '**gpm**' non si avvii regolarmente, anche se la configurazione predefinita corrisponde alla situazione reale. In tal caso è necessario riavviare manualmente il servizio nel modo appena mostrato.

Configurazione di X

La configurazione iniziale contenuta nel file '`/etc/xorg.conf`' è predisposta automaticamente se il file system è in sola lettura; diversamente potrebbe trattarsi di una configurazione generica VESA. In generale, se la configurazione non è stata determinata in modo automatico o comunque se non è soddisfacente, può essere conveniente modificarla con il comando '**nlnxrc x config**', operando in qualità di amministratore:

```
# nlnxrc x config [Invio]
```

La prima cosa che viene richiesto di specificare è l'adattatore grafico:

```
01:00.0 VGA compatible controller: nVidia Corporation NV5M64
| Please, select one of the following video adapters:      |
| .^(-)---------------------------------------------------. |
| |cyrix    /usr/lib/xorg/modules/drivers/cyrix_drv.so   | |
| |dummy    /usr/lib/xorg/modules/drivers/dummy_drv.so   | |
| |fbdev    /usr/lib/xorg/modules/drivers/fbdev_drv.so   | |
| |glint    /usr/lib/xorg/modules/drivers/glint_drv.so   | |
| |i128     /usr/lib/xorg/modules/drivers/i128_drv.so    | |
| |i740     /usr/lib/xorg/modules/drivers/i740_drv.so    | |
| |i810     /usr/lib/xorg/modules/drivers/i810_drv.so    | |
| |imstt    /usr/lib/xorg/modules/drivers/imstt_drv.so   | |
| |mga      /usr/lib/xorg/modules/drivers/mga_drv.so     | |
| |neomagic /usr/lib/xorg/modules/drivers/neomagic_drv.so| |
| |newport  /usr/lib/xorg/modules/drivers/newport_drv.so | |
| |nsc      /usr/lib/xorg/modules/drivers/nsc_drv.so     | |
| |nv       /usr/lib/xorg/modules/drivers/nv_drv.so      | |
| `v(+)---------------------------------------------------' |
|----------------------------------------------------------|
|              <  OK  >           <Cancel>                |
`----------------------------------------------------------'
```


Si può osservare, in alto, che viene suggerito il nome dell'adattatore grafico, in base a quanto riportato dal bus PCI. In questo caso, va scelta la voce '**nv**', portandovi sopra il cursore e premendo [*Invio*] per confermare.

La richiesta successiva riguarda la mappa della tastiera da usare durante il funzionamento grafico. Si deve specificare una mappa principale e una alternativa (la mappa principale deve essere latina):

```
.----------Select the X main keyboard-------------.
| Please, select one of the following keyboard    |
| maps, to be used as the MAIN one, It should be  |
| a latin keyboard map:                           |
| .---^(-)-------------------------------------. |
| |   hu          Hungarian                     | |
| |   is          Icelandic                     | |
| |   it          Italian                       | |
| |   la          Latin America                 | |
| |   lt          Lithuanian qwerty numeric     | |
| |   lt_std      Lithuanian azerty standard    | |
| |   lv          Latvian                       | |
| |   mt          Maltese                       | |
| |   no          Norvegian                     | |
| `---v(+)-------------------------------------' |
|-------------------------------------------------|
|            <  OK  >       <Cancel>             |
`-------------------------------------------------'
```

```
.--------Select the X alternate keyboard----------.
| Please, select one of the following keyboard    |
| maps, to be used as the alternate one:          |
| .---------------------------------------------. |
| |   us          previous                      | |
| |   us          U.S. English                  | |
| |   en_US       U.S. English w/ ISO9995-3     | |
| |   us_intl     U.S. English w/ deadkeys      | |
| |   al          Albanian                      | |
| |   ar          Arabic                        | |
| |   am          Armenian                      | |
| |   az          Azerbaidjani                  | |
| |   by          Belarusian                    | |
| |   be          Belgian                       | |
| |   ben         Bengali                       | |
| `---v(+)-------------------------------------' |
|-------------------------------------------------|
|            <  OK  >       <Cancel>             |
`-------------------------------------------------'
```

Alla fine si specifica la frequenza di scansione, la profondità di colori e la geometria. Questa è la scelta più delicata:

```
.-----------Screen frequencies and color depth-------------.
| Please, select one of the following screen frequencies,  |
| color depth and geometry combinations. The ranges with   |
| high frequencies are very dangerous for you screen if it |
| cannot handle them!                                      |
|                                                          |
| Horizontal kHz;    Vertical Hz; b/px; Geometry           |
| .^(-)---------------------------------------------------. |
| |30-80;  50-90;   24; auto       4:3 24 bit/pixel      | |
| |30-80;  50-90;   16; 1024x768   4:3 16 bit/pixel      | |
| |30-80;  50-90;   24; 1024x768   4:3 24 bit/pixel      | |
| |30-80;  50-90;   16; 1280x960   4:3 16 bit/pixel      | |
| |30-80;  50-90;   24; 1280x960   4:3 24 bit/pixel      | |
| |30-80;  50-90;   16; 1280x1024  5:4 16 bit/pixel      | |
| |30-80;  50-90;   24; 1280x1024  5:4 24 bit/pixel      | |
| |30-80;  50-90;   16; 1600x960   5:3 16 bit/pixel      | |
| |30-80;  50-90;   24; 1600x960   5:3 24 bit/pixel      | |
| |30-80;  50-90;   16; 1600x1200  4:3 16 bit/pixel      | |
| |30-80;  50-90;   24; 1600x1200  4:3 24 bit/pixel      | |
| `v(+)---------------------------------------------------' |
|----------------------------------------------------------|
|              <  OK  >           <Cancel>                |
`----------------------------------------------------------'
```

Se il risultato non è esattamente come si vorrebbe, dopo la configurazione guidata da '**nlnxrc**', si può anche ritoccare a mano il file '`/etc/xorg.conf`'. I punti salienti sono questi:

```
…
Section "Monitor"
    Identifier  "Generic Monitor"
```

```
    Option      "DPMS"
    HorizSync    30-80
    VertRefresh  50-90
EndSection
…
Section "Screen"
    Identifier  "Default Screen"
    Device      "Generic Adapter"
    Monitor     "Generic Monitor"
    DefaultDepth  24
    …
    SubSection "Display"
        Depth    16
        Modes    "1280x1024"
        ViewPort 0 0
    EndSubSection
    SubSection "Display"
        Depth    24
        Modes    "1280x1024"
        ViewPort 0 0
    EndSubSection
EndSection
…
```

Si osservi che quando si utilizza NLNX da DVD *live*, o comunque in una situazione in cui il file system principale è in sola lettura, se la grafica non si avvia automaticamente, occorre utilizzare il comando '**`startx`**' tradizionale. In tal caso, però, lo script '**`startx`**', prima dell'avvio, modifica il file '`/etc/xorg.conf`' al volo, facendo le stesse domande che farebbe '**`nlnxrc`**', come appena mostrato.

Stampa

«
Configurazione 99
Opzioni di avvio per la stampa 102

NLNX include un servente di stampa compatibile con il tipo BSD, dove il file '`/etc/printcap`' può essere configurato facilmente con l'aiuto di '**`nlnxrc`**'.

Configurazione

Con il comando '**`nlnxrc printer config`**' si rigenera il file '`/etc/printcap`', specificando una sola coda di stampa predefinita, che può corrispondere anche a un servizio presso un elaboratore remoto, inclusi quelli offerti da elaboratori MS-Windows:

```
# nlnxrc printer config [Invio]

 .Setup default printer filter type--.
 | Please, select one of the         |
 | following filter programs:        |
 | .-------------------------------. |
 | |   magicfilter  Magicfilter    | |
 | |   foomatic     Foomatic       | |
 | '-------------------------------' |
 |-----------------------------------|
 |       <  OK  >    <Cancel>       |
 '-----------------------------------'
```

Inizialmente viene chiesto di specificare quale programma usare per la gestione del filtro di stampa che si preferisce. Magicfilter è il più semplice, con lo svantaggio che l'elenco di stampanti è ridotto (con Magicfilter manca soprattutto la gestione attraverso il servente HP IJS), ma consente a chi sa usarlo di predisporre il proprio filtro personalizzato; Foomatic è il più adatto per i meno esperti, essendo fornito di un elenco molto grande di modelli di stampanti.

Quello che segue è l'elenco che può apparire dopo avere selezionato Magicfilter:

```
 .---------------Setup default printer type-----------------.
 | Please, select one of the following printer filters.     |
 | If you don't find your special printer, please try a     |
 | different name that might be compatible with the one     |
 | that you have.                                           |
 | .------------------------------------------------------. |
 | |bj10e        /etc/magicfilter/bj10e-filter            | |
 | |bj200        /etc/magicfilter/bj200-filter            | |
 | |bj600_draft  /etc/magicfilter/bj600_draft-filter      | |
 | |bj600        /etc/magicfilter/bj600-filter            | |
 | |bj610        /etc/magicfilter/bj610-filter            | |
 | |bj800_draft  /etc/magicfilter/bj800_draft-filter      | |
 | |bj800        /etc/magicfilter/bj800-filter            | |
 | |bjccolor     /etc/magicfilter/bjccolor-filter         | |
 | |cps300       /etc/magicfilter/cps300-filter           | |
 | 'v(+)------------------------------------------------' |
 |----------------------------------------------------------|
 |               <  OK  >           <Cancel>               |
 '----------------------------------------------------------'
```

Il menù seguente riguarda invece la scelta di Foomatic:

```
.----------------Setup default printer type-----------------.
| Please, select one of the following PPD.                 |
| .------------------------------------------------------. |
| |   HP-2000C-hpijs                          .          | |
| |   HP-2000C-pcl3                           .          | |
| |   HP-2500C-hpijs                          .          | |
| |   HP-2500CM-hpijs                         .          | |
| |   HP-2500CM-Postscript                    .          | |
| |   HP-2500C-pcl3                           .          | |
| |   HP-2563-lp2563                          .          | |
| |   HP-Business_Inkjet_1100-hpijs           .          | |
| |   HP-Business_Inkjet_1200-hpijs           .          | |
| |   HP-Business_Inkjet_2200-chp2200         .          | |
| |   HP-Business_Inkjet_2200-hpijs           .          | |
| '---v(+)-----------------------------------------------' |
|----------------------------------------------------------|
|               <  OK  >         <Cancel>                 |
'----------------------------------------------------------'
```

Nel caso di Foomatic, i nomi dei modelli di stampante contengono anche un'estensione che fa capire in che modo viene realizzato il filtro; pertanto, spesso sono disponibili filtri differenti per uno stesso modello. Si osservi che non è garantito che funzionino tutte le voci e può essere necessario fare dei tentativi.

Dopo avere specificato il nome del filtro da usare, si può indicare l'indirizzi IPv4 presso il quale si trova la stampante; se si tratta dell'indirizzo 127.0.0.1 o se si lascia vuoto il campo, si intende fare riferimento a una stampante locale. In alternativa, si può indicare un percorso per raggiungere una stampante condivisa di MS-Windows.

```
.----------------Printer location------------------.
| Please insert the printer location address; if   |
| the printer is local, just use "127.0.0.1" or    |
| leave it blank; if it is a BSD remote printer,   |
| put the IPv4 address; if it is a SMB (Windows)   |
| remote printer, enter the resource path, like:   |
| "//hostname/printer" (do not use backslash).     |
| .--------------------------------------------. |
| |127.0.0.1                                   | |
| '--------------------------------------------' |
|--------------------------------------------------|
|             <  OK  >       <Cancel>              |
'--------------------------------------------------'
```

Se si fa riferimento a una stampante locale (come appare nell'esempio), viene richiesto di indicare il file di dispositivo a cui corrisponde la stampante:

```
.---Setup default printer device-----.
| Please, select one of the          |
| following printer devices:         |
| .--------------------------------. |
| |   /dev/lp0      /dev/lp0       | |
| |   /dev/usb/lp0  /dev/usb/lp0   | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >     <Cancel>        |
'------------------------------------'
```

Quando si installa NLNX nel disco fisso, se questo non viene aggiornato automaticamente, tramite il protocollo DHCP, è necessario predisporre il file '`/etc/printcap`' con l'aiuto del comando '**`nlnxrc printer config`**', ritoccandolo eventualmente a mano in un momento successivo. Segue un esempio riferito alla stampa compatibile con un modello HP Laserjet generico, collegata alla prima porta parallela:

```
lp:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :if=/etc/magicfilter/laserjet-filter:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :sh:
```

Con il comando '**`nlnxrc printer config`**' è possibile configurare una sola stampante; se si vogliono gestire più stampanti, o semplicemente più code differenti, occorre modificare il file '`/etc/printcap`' a mano. L'esempio seguente riporta il caso di due code di stampa: quella predefinita riguarda una stampante locale HP Laserjet, mentre la coda successiva ('**`lp2`**') è rivolta a una stampante remota (192.168.1.254), senza specificare il tipo di filtro:

```
lp:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :if=/etc/magicfilter/laserjet-filter:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :sh:

lp2:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :mc#999:\
        :rp=lp:\
        :rm=192.168.1.254:\
        :sh:
```

Per approfondire l'argomento si può consultare il capitolo 27 dedicato alla stampa.

Va osservato che se si richiede di utilizzare una stampante condivisa, specificando un percorso del tipo '`//pc01/stampante`', tale percorso va usato utilizzando solo barre oblique normali, e non le barre inclinate nel modo opposto, come fa MS-Windows; inoltre, se il nome della stampante contiene spazi, al loro posto va usata la sequenza '**`%20`**'. Infine, va tenuto presente che la configurazione non si riflette sostanzialmente nel file '`/etc/printcap`', perché sono gli script '`/etc/script/lpr`' e '`/etc/script/lp`' che si occupano, in tal caso, di inviare il file da stampare (filtrato opportunamente) alla condivisione. Ciò significa anche che per raggiungere una stampante condivisa, non è possibile usare direttamente il comando '`/usr/bin/lpr`', ma bisogna avvalersi necessariamente degli script di NLNX.

```
.----------------Printer location------------------.
| Please insert the printer location address; if   |
| the printer is local, just use "127.0.0.1" or    |
| leave it blank; if it is a BSD remote printer,   |
| put the IPv4 address; if it is a SMB (Windows)   |
| remote printer, enter the resource path, like:   |
| "//hostname/printer" (do not use backslash).     |
| .--------------------------------------------. |
| |//docenti/stampante                         | |
| '--------------------------------------------' |
|--------------------------------------------------|
|             <  OK  >       <Cancel>              |
'--------------------------------------------------'
```

Quando si specifica un percorso di stampa riferito a una condivisione MS-Windows, viene richiesto di specificare eventualmente il nominativo dell'utente e la parola d'ordine per accedere al servizio. Naturalmente, se questo servizio non lo richiede, tali dati aggiuntivi non vanno inseriti.

```
.----------Printer user------------.
| Please insert the user to access |
| the remote printer               |
| "//docenti/stampante". If there  |
| is no user, just leave it blank. |
|                                  |
| .------------------------------. |
| |                              | |
| '------------------------------' |
|----------------------------------|
|       <  OK  >    <Cancel>       |
'----------------------------------'

.--------Printer password----------.
| Please insert the password to    |
| access the remote printer        |
| "//docenti/stampante" as user    |
| "daniele". If there is no        |
| password, just leave it blank.   |
| .------------------------------. |
| |                              | |
| '------------------------------' |
|----------------------------------|
|       <  OK  >    <Cancel>       |
'----------------------------------'
```

Opzioni di avvio per la stampa

Tra le opzioni di avvio di NLNX, ne sono previste alcune per la configurazione al volo della stampante. In particolare, '**`n_lpr_filter`**' si può usare sia per la stampa «normale» (BSD), sia per quella con le stampanti condivise di MS-Windows allo scopo di consentire l'indicazione del filtro da applicare ai file che vengono inviati alla stampante. Il nome del filtro deriva dal nome del file contenuto nella directory '`/etc/magicfilter/`', togliendo la terminazione '`-filter`', oppure dal nome del file PPD, togliendo l'estensione '`.ppd`' o '`.ppd.gz`'. Per esempio, volendo utilizzare il file '`ljet4-filter`' va scritto semplicemente il nome '**`ljet4`**', mentre volendo utilizzare il file '`HP-LaserJet_4-ljet4.ppd.gz`' va scritto solo il nome '**`HP-LaserJet_4-ljet4`**'. Segue la descrizione di alcuni esempi un po' più completi.

- `n_lpr_server=127.0.0.1 n_lpr_filter=laserjet`

  `n_lpr_filter=laserjet`

  Configura la stampa per una stampante locale, utilizzando un filtro di stampa per il tipo «laserjet» (corrispondente al file '`/etc/magicfilter/laserjet-filter`'. Il file di dispositivo corrispondente alla stampante locale viene individuato automaticamente, per esclusione.

- `n_lpr_server=172.21.254.254 n_lpr_filter=laserjet`

  Configura la stampa per una stampante di rete, all'indirizzo 172.21.254.254, utilizzando un filtro di stampa per il tipo «laserjet» (corrispondente al file '`/etc/magicfilter/laserjet-filter`'.

- `n_smp_prn=//pc77/stampante n_smb_prn_user=root n_lpr_filter=laserjet`

  Configura la stampa per una stampante di rete, condivisa da un elaboratore MS-Windows con il nome '`\\pc77\stampante`', autenticandosi come amministratore e senza fornire alcuna parola d'ordine. Il filtro di stampa è lo stesso visto negli esempi precedenti.

- `n_smp_prn=//pc77/stampante n_smb_prn_user=root` ↩
  `↪n_smb_prn_passwd=segreta n_lpr_filter=laserjet`

  Come nell'esempio precedente, specificando una parola d'ordine.

- `n_smp_prn=//192.168.0.7/stampante n_smb_prn_user=root` ↩
  `↪n_smb_prn_passwd=segreta n_lpr_filter=laserjet`

  Come nell'esempio precedente, indicando al posto del nome dell'elaboratore che offre la condivisione della stampante, il suo indirizzo IPv4.

- `n_max_pages=11`

  Fissa il numero massimo di pagine stampabili per volta.

Particolarità varie di NLNX

ACPI 103
Configurazione di Bash e script di sistema 104
Procedura di inizializzazione del sistema 104
Registri 106
Pianificazione dei processi con Cron 106
Dati variabili nel DVD *live* 107
Blocco delle funzioni di «nlnxrc» 107

`clamav-home` 106 `nlnx.config` 104 `nlnx.hardware` 104 `nlnx.last` 104 `nlnx.mixed` 104 `nlnx.network` 104 `nlnx.ro-fs` 104 `rc.local` 104 `syskogd` 106

Le funzionalità principali di NLNX sono gestite attraverso lo script '**`nlnxrc`**' ('`/etc/script/nlnxrc`') che può essere utilizzato anche senza argomenti, ottenendo così un menù delle funzionalità principali:

```
# nlnxrc [Invio]
```

Figura u26.1. Come si presenta lo script '**`nlnxrc`**' quando viene avviato senza argomenti, per ottenere il menù dei comandi disponibili. Le voci sono raggruppate in base al contesto per le quali possono essere più interessanti.

```
.-------------------------------NLNX--------------------------------.
| nlnxrc menu                                                        |
| Some functions are reserved to "root" or might be reserved for     |
| usage with read-write or read-only file system: see "nlnxrc --help"|
| for a contextual command list.                                     |
| .^(-)--------------------------------------------------------. |
| |user add              #rw Add a new user                     | |
| |user del              #rw Remove an old user                 | |
| |user passwd           $rw Change the user password           | |
| |user info             #rw Show info on a selected common user| |
| |home info             #rw Show info on a selected home directory| |
| |quota set             #rw Set user's disk quota              | |
| |nis-server-users edit #rw Edit NIS server allowed users      | |
| |print maxpages        #   Set max pages to print with lpr/lp| |
| |--------------------- ---------------------------------------| |
| |ssh hostkey           #   Make some new host keys            | |
| |ssh userkey           $   Make some new RSA keys for the user| |
| |--------------------- ---------------------------------------| |
| 'v(+)------------------------------------------------------------' |
|--------------------------------------------------------------------|
|                  <  OK  >           <Cancel>                       |
'--------------------------------------------------------------------'
```

Si osservi che all'inizio delle voci appaiono delle sigle: '**`$`**' indica una funzione disponibile a tutti gli utenti; '**`#`**' indica una funzione disponibile soltanto all'amministratore; '**`#ro`**' è una funzione disponibile soltanto all'amministratore durante il funzionamento con un file system in sola lettura (come nel DVD *live*); '**`#rw`**' indica una funzione che si può utilizzare solo come amministratore, quando il sistema è installato in un disco normale (con accesso in lettura e scrittura).

ACPI

Il kernel standard di NLNX include delle funzionalità ACPI. In diversi casi, queste funzionalità sono limitate o disabilitate del tutto, perché all'avvio viene passata al kernel l'opzione '**`acpi=strict`**' o '**`acpi=off`**', ma in tali situazioni si possono abilitare facilmente (inserendo all'avvio l'opzione '**`acpi=on`**', oppure modificando la configurazione di GRUB 1 con il file '`grub/menu.lst`', o di SYSLINUX con il file '`syslinux.cfg`', una volta installato NLNX).

Quando sono attive le funzionalità ACPI, è possibile controllare sommariamente il tempo di ritardo per lo spegnimento dei dischi, attraverso '**`nlnxrc`**':

```
# nlnxrc acpi spindown [Invio]
```

Se si vuole gestire lo spegnimento dei dischi in modo indipendente da quanto predisposto da NLNX (utilizzando il programma

'**hdparm**', con l'opzione '**-S**', a mano), bisogna accertarsi di eliminare il file '`/etc/nlnx/ACPI_STAND_BY_VALUE`', che altrimenti viene preso in considerazione all'avvio del sistema.

Configurazione di Bash e script di sistema

Il file '`/etc/profile`' è più articolato di quello standard. Vengono descritte le particolarità più significative.

- La variabile '**PATH**' tiene conto anche di quanto contenuto nelle directory '`/opt/*/bin/`' ed è prevista la directory '`/etc/script/`' per gli script che non appartengono allo standard della distribuzione GNU/Linux Debian. Il percorso della directory '`/etc/script/`' viene posto prima degli altri, in modo da avere la precedenza nella scelta di nomi uguali.
- L'invito della shell è costruito in modo da mostrare un indirizzo IPv4 utile per identificare il proprio elaboratore. L'indirizzo IPv4 viene ottenuto leggendo lo stato attuale della configurazione delle interfacce di rete; se l'indirizzo cambia, per aggiornare l'invito è necessario uscire e rientrare dalla sessione di lavoro.
- Sono previsti una serie di alias per i comandi '**rm**', '**cp**', '**mv**' e '**ln**', in modo da usare sempre l'opzione '**-i**'.
- Se l'utente che tenta di accedere non è l'amministratore e si tratta di un elaboratore che sembra essere dedicato alla condivisione delle utenze attraverso NFS e NIS, si fa in modo che la shell termini di funzionare, prima che l'utente possa avere a disposizione l'invito. Tuttavia, è possibile definire un elenco di utenti comuni che non sono sottoposti a questa limitazione, nel file '`/etc/nlnx/NIS_SERVER_ALLOWED_USERS`'.
- Se l'utente che tenta di accedere non è l'amministratore e il sistema operativo è installato in un file system usato in lettura e scrittura, l'accesso può essere sottoposto a un controllo preliminare sull'utilizzo del disco. Per attivare questa funzionalità in pratica, si utilizza il comando '**nlnxrc quota set**', con il quale viene richiesto di modificare il file '`/etc/nlnx/HOME_DISK_SPACE_ALLOWED`'. Il file ha una sintassi molto semplice: è composto da righe composte secondo il modello seguente:

  [*nominativo_utente*] : *n*

  Il numero che appare dopo i due punti ('**:**') è la quantità di byte a disposizione, mentre il nominativo utente, se non viene indicato rappresenta tutti gli utenti che nel file non sono stati specificati.

  Ogni volta che un utente accede al sistema, viene informato sullo stato dell'utilizzo del disco a partire dalla propria directory personale.

Procedura di inizializzazione del sistema

La procedura di inizializzazione del sistema prevede diversi script in più, come sintetizzato nella tabella successiva.

Tabella u26.2. Script aggiuntivi di NLNX, legati in qualche modo alla procedura di inizializzazione del sistema.

Script e collegamenti simbolici	Descrizione
'`/etc/script.functions/*`'	Funzioni generali di NLNX. Non vengono eseguiti da soli, ma incorporati in altri script, anche al di fuori della procedura di inizializzazione del sistema.
'`/etc/init.d/nlnx.ro-fs`' '`/etc/rcS.d/S19nlnx.ro-fs`'	È presente solo in un file system in sola lettura e serve a riprodurre la struttura necessaria delle directory '`/var/`', '`/home/`' e '`/etc/`' nel file system virtuale contenuto nel disco RAM, inoltre serve ad attivare la memoria virtuale nel caso riesca a trovare una partizione già prevista per questo.
'`/etc/init.d/nlnx.hardware`' '`/etc/rcS.d/S31nlnx.hardware`'	Configurazione riconducibile all'hardware e caricamento dei moduli relativi.
'`/etc/init.d/nlnx.config`' '`/etc/rcS.d/S32nlnx.config`'	Ripristino della configurazione particolare di NLNX e aggiornamento della configurazione sulla base delle opzioni di avvio.
'`/etc/init.d/nlnx.network`' '`/etc/rcS.d/S41nlnx.network`'	Configurazione della rete.
'`/etc/init.d/nlnx.mixed`' '`/etc/rcS.d/S98nlnx.mixed`'	È il raccoglitore di tutto quello che deve essere fatto alla fine del primo gruppo di script della procedura di avvio, contenuti nella directory '`/etc/rcS.d/`'.
'`/etc/init.d/nlnx.ipwatch`' '`/etc/rc[2|3|4|5].d/S20nlnx.ipwatch`'	Script di avvio e arresto del servizio IPwatch, con il quale il demone '**ipwatch**' verifica che il proprio indirizzo IPv4 non venga annunciato da altri nella rete locale. Tale servizio serve particolarmente per un elaboratore servente, il quale non deve risultare oscurato improvvisamente da un conflitto di indirizzi IP.
'`/etc/init.d/nlnx.watchdog`' '`/etc/rc[2|3|4|5].d/S98nlnx.watchdog`'	È uno script da adattare e utilizzare espressamente, perché altrimenti non viene avviato in modo predefinito. Il suo scopo è quello di accertarsi del funzionamento di certi servizi e, in mancanza di risposta, provvedere al loro riavvio. Per questo si avvale di altri script, denominati '`nlnx_watchdog_...`'.
'`/etc/init.d/nlnx.display_manager`' '`/etc/rc[2|3|4|5].d/S99nlnx.display_manager`'	Se si utilizza l'opzione di avvio '**n_auto_dm=1**', invece dell'accesso consueto, si ottiene un menù con cui selezionare uno degli utenti predefiniti, ciò attraverso questo script.

Script e collegamenti simbolici	Descrizione
‘/etc/init.d/nlnx.last’ ‘/etc/rc[2|3|4|5].d/S99nlnx.last’	Contiene ciò che va fatto quasi alla fine della procedura di avvio, pertanto si trova richiamato dalle directory ‘/etc/rc2.d/’, ‘/etc/rc3.d/’, ‘/etc/rc4.d/’ e ‘/etc/rc5.d/’; in particolare, questo script contiene la configurazione del tempo di ritardo per lo spegnimento dei dischi se le funzioni ACPI sono attive.
‘/etc/init.d/nlnx.mnt_fs’ ‘/etc/init.d/nlnx.mnt_hd’ ‘/etc/init.d/nlnx.mnt_md’ ‘/etc/init.d/nlnx.mnt_sd’ ‘/etc/init.d/nlnx.mnt_sr’	Ricostruzione dei punti di innesto a partire dalla directory ‘/mnt/’. Questi script vengono eseguiti da uDev e riguardano la procedura di inizializzazione del sistema in modo indiretto.

Il file ‘/etc/rc.local’ è a disposizione della personalizzazione, secondo la convenzione di diverse distribuzioni GNU/Linux.

Registri

Nella sua configurazione predefinita, NLNX prevede che il file ‘/var/log/syslog’ venga usato per accumulare una copia di tutti i messaggi che riguardano il registro di sistema (log); inoltre, prevede che la rotazione di questo file avvenga in modo tale da poter disporre sempre di almeno un anno di utilizzo dell’elaboratore. Per controllare la rotazione del file, è stato necessario modificare i file ‘/etc/cron.daily/sysklogd’ e ‘/etc/cron.weekly/sysklogd’, che nella distribuzione GNU/Linux Debian sono organizzati inizialmente per una sola settimana di dati. I file in questione sono modificati nel modo seguente:

```
…
cd /var/log
for LOG in ‘syslogd-listfiles‘
do
   if [ -s $LOG ]; then
      savelog -g adm -m 644 -u root -c 400 $LOG > /dev/null
   fi
done
…
```

```
…
cd /var/log
for LOG in ‘syslogd-listfiles --weekly‘
do
   if [ -s $LOG ]; then
      savelog -g adm -m 644 -u root -c 58 $LOG > /dev/null
   fi
done
…
```

Oltre al numero di copie più alto rispetto al solito, si deve osservare che i permessi consentono a tutti di leggere questi file. Infatti, ciò è voluto proprio per motivi didattici, allo scopo di consentire a tutti gli utenti lo studio del contenuto dei registri.

Pianificazione dei processi con Cron

Sono presenti alcuni script realizzati appositamente per NLNX, nelle directory usate da Cron per l’esecuzione pianificata dei processi. La tabella successiva riassume quelli più importanti.

Tabella u26.5. Script aggiuntivi di NLNX, relativi all’esecuzione periodica di processi.

Script	Descrizione
‘/etc/cron.01/nlnx-img’	Verifica se esiste il file (o il collegamento) ‘/opt/nlnx/nlnx.img’ e se è così, lo innesta (o reinnesta) come file-immagine, nella directory ‘/mnt/nlnx-img/’.
‘/etc/cron.daily/nlnx-clamav-home’	Avvia la scansione antivirus a partire dalla directory ‘/home/’. Se esiste la directory ‘/home/.virus/’, i file che risultano contaminati vengono spostati lì, altrimenti vengono cancellati.
‘/etc/cron.daily/nlnx-logfilter’	Rielabora il file ‘/var/log/syslog.0’, selezionando alcune informazioni e producendo i file ‘/var/log/access.***n***.log’, ‘/var/log/users.***n***.log’ e ‘/var/log/admin.***n***.log’. Il numero ***n*** è costituito dall’anno e dal giorno dell’anno in cui ha luogo l’elaborazione.
‘/etc/cron.daily/nlnx-quotacheck’	Ripristina il controllo delle quote di utilizzo del disco.
‘/etc/cron.daily/time-align’	Riallinea l’orologio interno con il riferimento presente nella configurazione.
‘/etc/cron.daily/nlnx-webalizer’	Esegue il comando ‘**nlnxrc webalizer update**’, con il quale si aggiorna, se c’è, la statistica degli accessi al servente HTTP locale.

Le directory usate per gli script di Cron sono in numero maggiore rispetto alle distribuzioni GNU/Linux comuni. In particolare si trovano le directory ‘/etc/cron.***nn***h’, dove ***nn*** può essere ‘**01**’, ‘**02**’, ‘**03**’, ‘**04**’, ‘**06**’, ‘**08**’ o ‘**12**’, a indicare ognuna una scansione di ***nn*** ore.

Nelle directory degli script di Cron, è possibile predisporre un file denominato ‘cron_local’, per usi personali. Questo file viene ignorato quando si va a produrre un nuovo DVD *live* di NLNX.

Dati variabili nel DVD *live*

Quando NLNX funziona da un file system in sola lettura (come nel caso del DVD *live*), il contenuto di alcune directory, come ‘/etc/’ e ‘/var/’, riguarda in realtà ciò che si trova a partire da ‘/ramdisk/’, che a sua volta è il punto di innesto di un disco RAM. In tal modo, il contenuto di queste directory può essere modificato, anche se solo temporaneamente; ovviamente, per risparmiare memoria, la maggior parte dei file presenti a partire da ‘/ramdisk/’ è costituita da collegamenti simbolici ad altri file che, in tale contesto, appaiono a partire da ‘/RW-FS/…’ e come tali non sono modificabili. Per poter modificare tali file occorre cancellare i collegamenti simbolici relativi, sostituendoli con una copia del file a cui questi puntano.

Blocco delle funzioni di «nlnxrc»

Una volta configurate alcune funzionalità del sistema operativo, attraverso l’uso di ‘**nlnxrc**’, oppure a mano, può essere opportuno fare in modo che certi comandi di ‘**nlnxrc**’ non funzionino, per evitare che quanto fatto con cura venga annullato con una piccola disattenzione. In altri termini, una volta sistemata la configurazione, per evitare errori è possibile impedire a ‘**nlnxrc**’ di ritornarci sopra,

bloccando selettivamente alcuni comandi. Per fare questo occorre creare a mano il file ‘`/etc/nlnx/NLNXRC_DISABLED_COMMANDS`’, contenente direttive molto semplici, come quelle dell’esempio seguente:

```
…
#
# Configurazione già fatta, da non modificare più.
#
printer:config
network:config
…
```

Come si può intuire, il cancelletto (‘`#`’) introduce un commento che viene ignorato, mentre le direttive richiamano dei comandi di ‘**`nlnxrc`**’. Da quello che si vede, sono bloccati i comandi: ‘**`nlnxrc printer config`**’ e ‘**`nlnxrc network config`**’.

Figura u26.7. Il messaggio con cui ‘**`nlnxrc`**’ spiega l’impossibilità di eseguire il comando ‘**`nlnxrc printer config`**’.

```
.-----------------------------------------------.
| The command "nlnxrc printer config" is        |
| disabled, as configured inside                |
| "/etc/nlnx/NLNXRC_DISABLED_COMMANDS"!         |
|                                               |
|-----------------------------------------------|
|                   <  OK  >                    |
'-----------------------------------------------'
```


Servizi di rete vari, secondo l’organizzazione di NLNX

Nomi a dominio 109
DHCP 109
TFTP e PXE 110
HTTP 110
Proxy HTTP 110
File di registrazioni 111
Orologio di riferimento 111
Servizi da avviare manualmente 111
Utilizzo dello scanner 112

`nlnx.network` 109

NLNX offre diversi servizi, sia localmente, sia attraverso la rete. Alcuni di questi sono attivabili anche durante il funzionamento da unità in sola lettura. Per leggere le sezioni successive, si tengano come riferimento anche le tabelle u16.2, u16.4, u16.10 e u16.5.

Nomi a dominio

I file di configurazione di BIND, per la risoluzione dei nomi, sono collocati tutti nella directory ‘`/etc/bind/`’ e le zone di competenza, nell’impostazione iniziale, si riferiscono all’indirizzo 127.0.0.1.

Il nome *nlnx* viene indicato nel file ‘`/etc/hosts`’, come sinonimo di *localhost*, per garantire il funzionamento di alcuni programmi (si veda il capitolo 33 a proposito della configurazione di un servizio DNS con Bind). Se all’elaboratore si vuole attribuire un nome differente da ‘**`nlnx`**’, è necessario correggere il file ‘`/etc/hosts`’, in modo che contempli anche questo come sinonimo di 127.0.0.1.

Il file ‘`/etc/resolv.conf`’ iniziale è configurato in modo da interrogare l’elaboratore locale, ma con la presenza di altri indirizzi di serventi DNS di fornitori ben conosciuti, i quali potrebbero tornare utili in caso di emergenza, o anche solo per conoscenza. Tuttavia, in presenza di una configurazione automatica della rete, questo file viene aggiornato, perdendo le informazioni originarie.

DHCP

NLNX dispone di un cliente DHCP che viene usato in modo predefinito per la configurazione della rete locale.

L’utilizzo di un servizio DHCP può essere molto utile quando si usa NLNX da unità in sola lettura, senza un’organizzazione particolare del lavoro. Ma quando si installa NLNX in una serie di elaboratori, è preferibile avere un’attribuzione precisa degli indirizzi, anche senza dipendere necessariamente da un servizio DHCP.

NLNX utilizza diverse «opzioni» DHCP per la propria configurazione, secondo lo schema della tabella seguente. Si osservi comunque che le direttive utilizzate sono più numerose di quelle descritte qui.

Opzione DHCP	Utilizzo

`subnet-mask`
`broadcast-address`
`routers`
	Queste opzioni, assieme all’informazione sull’indirizzo IPv4, consentono di configurare l’interfaccia di rete e gli instradamenti.

`domain-name-servers`	Permette di modificare automaticamente il file ‘`/etc/resolv.conf`’.
`time-servers` `ntp-servers`	Una di queste due opzioni permette di sincronizzare l’orologio locale con il protocollo RDATE o NTP.
`root-path`	La disponibilità di questa opzione fa sì che si tenti di innestare la directory ‘`home/`’ remota.

Opzione DHCP	Utilizzo

`nis-domain`
 	
`nis-servers`
	Queste due opzioni consentono di utilizzare il protocollo NIS per la gestione condivisa delle utenze.

`lpr-servers`	Permette di modificare il file '`/etc/printcap`', in modo da usare una stampante remota.
`log-servers`	Permette di modificare il file '`/etc/syslog.conf`', in modo da inviare una copia del registro del sistema a un elaboratore remoto.

Il servizio DHCP viene fornito solo per l'interfaccia di rete che risulta essere "interna"; tuttavia, dal momento che si creano delle complicazioni quando si utilizza un elaboratore con una sola interfaccia, che però opera con più indirizzi, l'avvio del servizio viene gestito direttamente dallo script '`/etc/init.d/nlnx.network`'. Per questo, lo script '`/etc/init.d/dhcpd3-server`' non è in grado di avviare o di riavviare il servizio, ma solo di fermarlo.

Si veda anche la sezione u33 per una trattazione più ampia del contesto in cui si inserisce l'utilizzo del DHCP con NLNX.

TFTP e PXE

NLNX dispone di un servente TFTP per la condivisione della directory '`/var/lib/tftpboot/`', allo scopo di consentire l'avvio dalla rete, attraverso il protocollo PXE. L'avvio attraverso il protocollo PXE si avvale di PXELINUX, costituito in pratica dal file '`/var/lib/tftpboot/pxelinux/pxelinux.0`' e dalla sua configurazione contenuta nella directory '`/var/lib/tftpboot/pxelinux/pxelinux.cfg/`'; gli altri file contenuti nella directory '`/var/lib/tftpboot/pxelinux/`' vengono caricati attraverso il protocollo PXE, in base alla configurazione predisposta con PXELINUX.

Si veda anche il capitolo u28 per una trattazione più ampia dell'uso di questi protocolli con NLNX.

HTTP

È disponibile il servente HTTP Mathopd. Allo scopo di semplificare il lavoro, si possono eseguire i programmi CGI se il nome del file corrispondente ha un'estensione del tipo '`.cgi`', '`.pl`' o '`.sh`'; inoltre, se è installato l'interprete PHP, i file con estensione '`.php`', '`.phtml`' o '`.pht`' vengono trattati attraverso questo sistema.

Si veda anche la sezione u29 a questo proposito.

Proxy HTTP

NLNX include un servizio proxy HTTP, costituito da OOPS, che può essere usato in modo trasparente. In generale, OOPS si avvia automaticamente solo quando NLNX è stato installato secondo una modalità normale (per cui il file system viene usato in lettura e scrittura), altrimenti dal DVD *live* o da altra unità in sola lettura, è possibile avviare il servizio attraverso il comando seguente:

```
# /etc/init.d/oops start [Invio]
```

La configurazione predefinita prevede l'ascolto presso la porta 3128; inoltre si fa a meno di usare file su disco per conservare le pagine e le altre risorse già visitate. Il funzionamento di un proxy HTTP e di OOPS in particolare è descritto nella sezione 42.2.

La configurazione di OOPS viene ripristinata dallo script '`/etc/init.d/nlnx.config`', a ogni avvio del sistema, da una copia di sicurezza. Ciò permette di non perdere inavvertitamente la configurazione standard quando si aggiorna il programma; tuttavia, se NLNX è stato installato nel disco fisso e si vuole intervenire nella configurazione di OOPS, occorre ricordarsi di fare le stesse modifiche anche nel file '`/etc/oops/oops.cfg.nlnx`', il quale rappresenta tale copia di sicurezza.

File di registrazioni

È possibile modificare facilmente la configurazione del registro di sistema allo scopo di inviare una copia di ciò che accade presso un elaboratore unico. Invece di modificare direttamente il file '`/etc/syslog.conf`', si può intervenire con il comando seguente:

```
# nlnxrc log-server config [Invio]

  .--------Log server----------.
  | Please specify the log     |
  | server to send system log: |
  | .------------------------. |
  | |                        | |
  | '------------------------' |
  |----------------------------|
  |   <  OK  >    <Cancel>     |
  '----------------------------'
```

Supponendo di voler inviare una copia all'elaboratore 192.168.1.254:

```
192.168.1.254 OK
```

Orologio di riferimento

È possibile sincronizzare il proprio elaboratore con un altro, che offra il servizio NTP o RDATE. Per questo è sufficiente usare il comando '**`nlnxrc date config`**':

```
# nlnxrc date config [Invio]

  .-------Rdate server---------.
  | Please specify the date    |
  | server you trust:          |
  | .------------------------. |
  | |                        | |
  | '------------------------' |
  |----------------------------|
  |   <  OK  >    <Cancel>     |
  '----------------------------'
```

```
pool.ntp.org OK
```

Se la configurazione riguarda una copia installata, oppure se viene salvata in qualche modo, all'avvio successivo l'allineamento dell'orologio avviene in modo automatico.

Per la sincronizzazione viene tentato un accesso attraverso '**`ntpdate`**', il quale prevede già una propria configurazione nel file '`/etc/default/ntpdate`'. In questo file conviene evitare l'indicazione di un servente NTP, perché il meccanismo usato da NLNX non lo richiede. Infatti, durante l'avvio del sistema, NLNX interroga l'indirizzo indicato nel modo mostrato sopra, attraverso il protocollo NTP, ma se questo fallisce riprova con il protocollo RDATE. Dal momento che queste operazioni richiedono un certo tempo, sono eseguite sullo sfondo, senza sospendere temporaneamente l'esecuzione dello script in cui sono contenute. Ciò ha il vantaggio di non rallentare l'avvio quando l'elaboratore non ha un accesso alla rete esterna; se invece si specifica un servente NTP nel file '`/etc/default/ntpdate`', da un lato si esegue per due volte la stessa funzione, dall'altro si ottiene una pausa inutile all'avvio se il servente NTP non può essere raggiunto.

Servizi da avviare manualmente

NLNX dispone di un servente per il protocollo SSH (Secure Shell), che però non viene avviato automaticamente con il sistema operativo, durante il funzionamento da unità in sola lettura (come i DVD), se la parola d'ordine per accedere in qualità di utente amministratore è rimasta quella predefinita. Come già descritto, anche la condivisione dei dati attraverso il protocollo NFS non è attiva in modo predefinito se si sta usando il sistema da unità in sola lettura. Per attivare questi servizi, rispettivamente, si può intervenire nel modo seguente:

```
# /etc/init.d/ssh start [Invio]

# /etc/init.d/nfs-kernel-server start [Invio]
```

Oltre ai servizi elencati, che, per motivi di sicurezza, non vengono avviati automaticamente da unità in sola lettura, anche altri non lo sono, per evitare di appesantire inutilmente il funzionamento, oppure perché il contesto richiede che non lo siano. Nella tabella successiva ne vengono riepilogati alcuni.

Tabella u27.4. Alcuni servizi che non vengono avviati automaticamente durante il funzionamento da unità in sola lettura.

Script	Descrizione
‘`/etc/init.d/ssh`’	Il servente per il protocollo SSH (Secure Shell) non viene avviato automaticamente per motivi di sicurezza, se la parola d’ordine dell’utente ‘**`root`**’ è quella predefinita per NLNX.
‘`/etc/init.d/nfs-kernel-server`’	Il servizio NFS non viene attivato automaticamente per motivi di sicurezza.
‘`/etc/init.d/nis`’	Il servente NIS non viene attivato, mentre, in condizioni normali il cliente NIS potrebbe essere attivato automaticamente, attraverso le informazioni ottenute dal cliente DHCP.
‘`/etc/init.d/oops`’	Proxy HTTP.
‘`/etc/init.d/anacron`’ ‘`/etc/init.d/cron`’	L’avvio di processi temporizzati non ha senso in un sistema basato su file system in sola lettura.

Utilizzo dello scanner

NLNX prevede la presenza di SANE per la gestione dello scanner. Per gli scanner che vengono riconosciuti automaticamente non ci sono problemi di utilizzo, inoltre è prevista una configurazione predefinita di SANE, tale da concedere l’accesso attraverso la rete, purché si tratti di indirizzi privati o comunque locali.

Per accedere a uno scanner remoto, è necessario intervenire nel file di configurazione ‘`/etc/sane.d/net.conf`’ di ogni nodo cliente; tuttavia, se si usa il DHCP, lo script che si occupa della configurazione dinamica aggiorna questo file inserendo tutti gli elaboratori che risultano fornire qualche servizio (anche se diverso), considerando che uno scanner di rete potrebbe essere collocato in uno di quelli.

In pratica, se si vuole usare il DHCP e si intende predisporre uno scanner di rete, conviene collocare questo presso lo stesso elaboratore che funge già da servente di stampa, oppure quello che offre il servizio NIS, oppure anche quello che si usa per accumulare il registro di sistema degli elaboratori appartenenti alla rete locale.

PXE per l’avvio di un elaboratore senza disco fisso

PXE 113
NLNX in un file system di rete 114
Controllo dell’avvio di sistemi locali 115

Con NLNX è possibile gestire l’avvio di elaboratori remoti, privi di disco fisso, principalmente per avviare un sistema completo, in un file system remoto, in sola lettura, funzionante in modo analogo a quello del DVD *live*. La configurazione di questo tipo di servizio riguarda il protocollo DHCP, ma può essere generata in modo guidato attraverso ‘**`nlnxrc`**’.

Tabella u16.4. Script ‘**`nlnxrc`**’: configurazione del servizio DHCP.

Comando	Descrizione

`nlnxrc dhcp-server config`
 	
`nlnxrc dhcp-server unconf`
	Attiva o disattiva il funzionamento in qualità di servente DHCP.

`nlnxrc dhcp-server edit`	Modifica la configurazione del servente DHCP, intervenendo nel file di configurazione in modo libero.

Va comunque osservato che il sistema di avvio remoto, organizzato per NLNX, potrebbe servire anche per avviare attraverso la rete degli elaboratori che dispongono localmente di una copia di NLNX, sia in una partizione propria, sia in un file-immagine, il quale potrebbe essere stato organizzato in sola lettura o in lettura e scrittura.

PXE

Un sistema NLNX installato secondo le modalità descritte in altri capitoli, dispone della directory ‘`/var/lib/tftpboot/`’, accessibile attraverso il protocollo TFTP (lo si vede nel file ‘`/etc/inetd.conf`’). A partire da questa directory si articolano altre directory e file che servono all’avvio di un sistema remoto, utilizzando inizialmente il protocollo PXE.

Per attivare il protocollo PXE si utilizza PXELINUX che fa parte in generale di SYSLINUX. PXELINUX richiede il caricamento e l’esecuzione di un piccolo programma, ‘`pxelinux.0`’, con il quale viene letto un menù, conforme al formato usato dai vari sistemi di SYSLINUX. Precisamente, in base alla configurazione prevista per NLNX, attraverso il protocollo DHCP e TFTP, l’elaboratore remoto carica ed esegue il file ‘`pxelinux/pxelinux.0`’;[1] successivamente questo programma carica il menù ‘`pxelinux/pxelinux.cfg/default`’.[2] Il file ‘`pxelinux/pxelinux.cfg/default`’ viene prodotto automaticamente da ‘**`nlnxrc`**’, quando lo si usa per configurare il servente DHCP.

Il menù contenuto nel file ‘`pxelinux/pxelinux.cfg/default`’ offre diverse tipologie di avvio remoto. In generale è possibile selezionare la voce ‘**`net`**’ che si riferisce all’avvio di un sistema NLNX su disco remoto e in sola lettura (analogamente al caso di un sistema su DVD *live*); in alternativa sono disponibili le voci ‘**`menu`**’ e ‘**`omenu`**’, il cui comportamento è esattamente uguale a quello delle stesse voci disponibili in un DVD, pertanto con queste è possibile avviare sia un sistema locale, sia un sistema di emergenza, per la manutenzione. È anche disponibile la voce ‘**`mem`**’, per avviare Memtest86+.

```
This is the NLNX PXE boot loader. Please insert a name and
then press [Enter].

    rescue              Generic rescue system
    diskless            Generic diskless system
    nowayout            Diskless network limited
    telnet              Diskless telnet server
    mem                 Memory check
```

Se non si seleziona alcuna voce, ma si preme ugualmente [*Invio*], viene tentato l'avvio dal disco fisso locale. Così si possono configurare gli elaboratori in modo che la prima voce di avvio (del *firmware*, ovvero del BIOS) tenti l'uso della rete, perché se non si esegue una selezione si passa automaticamente al disco locale.

Questa scelta consente, per esempio, di installare una copia di NLNX in un file-immagine ospitato nel file system locale di un altro sistema operativo, senza doversi prendere cura del suo avvio, che così avverrebbe semplicemente attraverso la rete.

NLNX in un file system di rete

Se gli elaboratori remoti, senza disco fisso, dispongono di memoria centrale sufficiente, oppure hanno un piccolo disco fisso con una partizione per lo scambio della memoria, è possibile fare in modo che si avvii un sistema completo, con le stesse facoltà di un NLNX avviato da un DVD *live*, ovvero con un file system principale in sola lettura.

Per realizzare questo obiettivo si installa una copia del file-immagine '`nlnx.img`' nella directory '`/opt/nlnx/`'.

Oltre alla copia, occorre verificare la configurazione del file '`/etc/exports`', il quale deve consentire un accesso in lettura a tale gerarchia, lasciando all'utente '**`root`**' tutti i suoi privilegi, come nell'esempio seguente che va adattato eventualmente alla propria rete locale:

```
…
/opt/nlnx   172.16.0.0/12(async,ro,no_root_squash,↩
↪nohide,subtree_check)
…
```

Avendo sistemato questo, con l'ausilio di '**`nlnxrc`**' si va a completare la configurazione, cosa che comporta anche la **copia dei file dei kernel e dei dischi RAM iniziali** corretti:[3]

```
# nlnxrc dhcp-server config [Invio]
```

```
.----------------------DHCP range-------------------------.
| Current "/etc/dhcp3/dhcpd.conf" file might be set as it  |
| follows:                                                 |
| ...                                                      |
| subnet 172.21.0.0 netmask 255.255.0.0 {                  |
| #  range 172.21.254.100 172.21.254.199;                  |
|   option broadcast-address 172.21.255.255;               |
|   option routers 172.21.254.254;                         |
|   option domain-name-servers 172.21.254.254;             |
|   option time-servers pool.ntp.org;                      |
|   option ntp-servers pool.ntp.org;                       |
|   option root-path "172.21.254.254:/opt/nlnx";           |
|   option nis-domain "nis.nano";                          |
|   option nis-servers 172.21.254.254;                     |
|   option lpr-servers 172.21.254.254;                     |
|   option log-servers 172.21.254.254;                     |
|   ...                                                    |
| }                                                        |
|                                                          |
| Please insert or confirm the DHCP address range:         |
| .--------------------------------------------------.     |
| |172.21.254.100 172.21.254.199                     | |
|-'--------------------------------------------------'-|
|               <  OK   >          <Annulla>               |
'----------------------------------------------------------'
```

Si ipotizza di voler utilizzare dinamicamente gli indirizzi da 172.21.1.100 a 172.21.1.199:

[*Canc*] [*Canc*] …

`172.21.1.100 172.21.1.199` OK

```
.--------------DHCP server configuration----------------.
| Is the following configuration correct?                |
|                                                        |
| ddns-update-style none;                                |
| option option-128 code 128 = string;                   |
| option option-129 code 129 = text;                     |
| subnet 172.21.0.0 netmask 255.255.0.0 {                |
|   range 172.21.1.100 172.21.1.199;                     |
|   option broadcast-address 172.21.255.255;             |
|   option routers 172.21.254.254;                       |
|   option domain-name-servers 172.21.254.254;           |
|   option time-servers pool.ntp.org;                    |
|   option ntp-servers pool.ntp.org;                     |
|   option root-path "172.21.254.254:/opt/nlnx";         |
|   option nis-domain "nis.nano";                        |
|   option nis-servers 172.21.254.254;                   |
|   option lpr-servers 172.21.254.254;                   |
|   option log-servers 172.21.254.254;                   |
|   use-host-decl-names      on;                         |
|   filename "/pxelinux/pxelinux.0";                     |
| }                                                      |
|--------------------------------------------------------|
|              < Yes >          < No  >                 |
'--------------------------------------------------------'
```

Se la configurazione proposta è quella che si desidera, si può confermare:

YES

L'installazione del file-immagine '`nlnx.img`' rappresenta un metodo semplice, pratico ed efficace, per mettere a disposizione NLNX in rete. È sufficiente che nella directory '`/opt/nlnx/`' sia presente tale file con il nome '`nlnx.img`':

```
# cp nlnx.img /opt/nlnx/ [Invio]
```

Controllo dell'avvio di sistemi locali

La disponibilità di un elaboratore con NLNX che offre il servizio di avvio tramite PXE, come descritto in questo capitolo, può essere utile anche per avviare degli elaboratori remoti, i quali dispongono sì del sistema operativo, ma non della possibilità di avviarlo. In tal caso, dal menù di PXE va scelta la voce '**`menu`**', oppure '**`auto`**', ma nessuna delle due è quella predefinita.

Eventualmente, se il servizio PXE dovesse servire soltanto per avviare elaboratori con un proprio NLNX locale, si potrebbe intervenire manualmente nella configurazione di PXELINUX nel file '`pxelinux/pxelinux.cfg`'.

È il caso di rammentare che la possibilità di avviare degli elaboratori remoti diventa molto importante quando NLNX è stato installato lì in un file-immagine, ospitato nel file system di un altro sistema operativo. In tal caso, quel file-immagine deve chiamarsi '`nlnx.img`' ed essere collocato nella radice di quel file system.

[1] Il percorso assoluto è '`/var/lib/tftpboot/pxelinux/pxelinux.0`'.

[2] Il percorso assoluto è '`/var/lib/tftpboot/pxelinux/pxelinux.cfg/default`'.

[3] Il sistema che viene installato nella directory '`/opt/nlnx/`', per funzionare in sola lettura attraverso la rete, ha un proprio insieme di moduli e una propria coppia di kernel, potenzialmente differenti rispetto al sistema complessivo che lo ospita. Pertanto, il sistema di avvio attraverso la rete, con il protocollo PXE, deve utilizzare questi kernel e i dischi RAM iniziali relativi.

Impedire la pubblicazione di file personali 117
Programmi CGI 117
PHP 117
Webalizer 118
MRTG 118
Accessibilità dei registri (log) 119
Documentazione interna 119
Esercitazioni con il linguaggio HTML 119

`public_html/` 117

NLNX offre un servente HTTP, costituito precisamente da Mathopd che si configura con il solo file '`/etc/mathopd.conf`'. In base alla configurazione predefinita di NLNX, il servente HTTP consente la pubblicazione di documenti a partire dalla directory '`/var/www/`', dove il file '`index.html`' va sostituito o semplicemente eliminato, e dalla directory '`~/public_html/`' di ogni utente.

La configurazione può essere modificata, ma per questo occorre intervenire manualmente nel file '`/etc/mathopd.conf`'.

Tabella u16.10. Script '**`nlnxrc`**': funzionalità relative a servizi offerti tramite il servente HTTP.

Comando	Descrizione
`nlnxrc mrtg config` `nlnxrc mrtg unconf`	Configura o disabilita le statistiche di utilizzo delle interfacce di rete, attraverso MRTG.
`nlnxrc webalizer update`	Aggiorna le statistiche di accesso al servente HTTP, prodotte con Webalizer.

Impedire la pubblicazione di file personali

NLNX è organizzato per poter soddisfare le esigenze dell'attività didattica in una scuola media superiore. A tale proposito, il fatto che gli utenti possano pubblicare dei file utilizzando la directory '`~/public_html/`' può essere un problema, in quanto gli studenti potrebbero così passarsi facilmente dei dati.

La configurazione predefinita di NLNX è tale per cui le directory personali degli utenti hanno i permessi 1771_8, pari a '**`rwxrwx--t`**', appartenendo all'utente '**`root`**' e al gruppo privato dell'utente rispettivo. In tal modo, gli utenti non hanno la facoltà di modificare i permessi della propria directory personale e non possono cancellare file o directory che non appartengono a loro stessi. Così facendo, per impedire che gli utenti possano pubblicare qualcosa attraverso il protocollo HTTP, è sufficiente che l'amministratore crei le directory '`~/public_html/`', appartenenti a se stesso, ma senza alcun permesso di accesso, lettura o modifica. Gli utenti, non potendo cancellare queste directory, non possono nemmeno collocarci dei file per la mancanza dei permessi necessari.

Programmi CGI

In base alla configurazione predefinita del servente HTTP, i file che hanno estensione '`.cgi`', '`.pl`' e '`.sh`', vengono eseguiti in qualità di programmi CGI (*Common gateway interface*). A titolo di esempio, nella directory '`/var/www/cgi/`' sono disponibili alcuni file di questo tipo; per visualizzarli occorre accedere all'indirizzo *http://**nodo**/cgi/*.

PHP

NLNX dispone normalmente dell'interprete PHP. Perché intervenga l'elaborazione di tale interprete è sufficiente che i file abbiano l'estensione '`.php`', oppure '`.phtml`' o '`.pht`'. Eventualmente, nella directory '`/var/www/php/`' (ovvero *http://**nodo**/php/*) si trova il fi-

le ‘`test.php`’ con il quale è possibile verificarne il funzionamento e conoscere le estensioni disponibili.

Webalizer

Attraverso lo script ‘`/etc/cron.daily/nlnx-webalizer`’ si ottiene l’aggiornamento quotidiano delle statistiche di accesso al servente HTTP. Se Mathopd è rimasto alla sua configurazione standard di NLNX, sono accessibili all’indirizzo *http://**nodo**/access/*. Inoltre, all’indirizzo *http://**nodo**/filter/* sono accessibili le statistiche degli accessi all’esterno, come contabilizzato da OOPS.

Per eseguire manualmente l’aggiornamento di queste statistiche si può usare il comando seguente:

```
# nlnxrc webalizer update [Invio]
```

MRTG

NLNX dispone di un servizio SNMP preconfigurato per l’accesso in sola lettura, in modo da consentire di ottenere le informazioni sul traffico che attraversa le interfacce di rete locali. Inoltre è disponibile MRTG, con il quale è possibile elaborare le statistiche di tale traffico all’indirizzo *http://**nodo**/mrtg/*. Tuttavia, MRTG va configurato in modo da fare riferimento alle interfacce di rete esistenti effettivamente. Va usato il comando seguente per fissare tale configurazione:

```
# nlnxrc mrgt config [Invio]
```

```
.------------------SNMP agent---------------------.
| Please insert a router SNMP agent address with  |
| community, like this: "COMMUNITY@ROUTER"        |
| When you have finished, select the "cancel"     |
| button.                                         |
| .---------------------------------------------. |
| |public@localhost                             | |
| ‘---------------------------------------------’ |
|-------------------------------------------------|
|           <  OK   >      <Cancel>               |
‘-------------------------------------------------’
```

OK

Dopo l’indicazione dell’elaboratore locale, si possono aggiungere altri router, come nell’esempio successivo. Al termine si deve annullare per concludere la configurazione.

```
.------------------SNMP agent---------------------.
| Please insert a router SNMP agent address with  |
| community, like this: "COMMUNITY@ROUTER"        |
| When you have finished, select the "cancel"     |
| button.                                         |
| .---------------------------------------------. |
| |public@192.168.1.254                         | |
| ‘---------------------------------------------’ |
|-------------------------------------------------|
|           <  OK   >      <Cancel>               |
‘-------------------------------------------------’
```

OK

...

CANCEL

```
.Clean MRTG from previous data--.
| Should I remove previous data |
| inside "/var/www/mrtg/"?      |
|                               |
|-------------------------------|
|     <  Yes >   <  No  >       |
‘-------------------------------’
```

YES

L’aggiornamento delle statistiche, in base alla configurazione prodotta, avviene attraverso uno script avviato da Cron.

Accessibilità dei registri (log)

In condizioni normali, principalmente per motivi didattici, i file del registro di sistema (log) e altri file analoghi contenuti a partire dalla directory ‘`/var/log/`’, sono accessibili a tutti gli utenti. In tal caso (se ci sono effettivamente i permessi di accesso in lettura per tutti), è possibile leggere il contenuto di questi registri anche attraverso un programma CGI, che consente l’accesso da parte di utenti di reti private (127.0.0.1, 10.0.0.0/8, 172.16.0.0/12 e 192.168.0.0/16): *http://**nodo**/cgi-bin/var_log*.

Se si accede all’indirizzo *http://**nodo**/cgi-bin/var_log* si ottiene la lista del contenuto della directory ‘`/var/log/`’ dell’elaboratore corrispondente al nome o all’indirizzo indicato (il «nodo»); se si aggiunge la stringa *?**percorso***, si intende visualizzare il contenuto del percorso ‘`/var/log/`***percorso**’. Per esempio, per visualizzare il contenuto del file ‘`/var/log/syslog`’, si deve usare l’indirizzo *http://**nodo**/cgi-bin/var_log?syslog*, mentre per leggere il contenuto del file ‘`/var/log/oops/access.log`’, si deve usare l’indirizzo *http://**nodo**/cgi-bin/var_log?oops/access.log*.

L’accessibilità a questi file dipende dai loro permessi, tenendo conto che il programma CGI funziona con gli stessi privilegi del servente HTTP.

Documentazione interna

Oltre ai servizi menzionati nel capitolo, attraverso il servente HTTP è disponibile della documentazione interna: le pagine di manuale, la documentazione Info (se installata) e *a2*. Nella tabella successiva sono riepilogati questi e gli altri indirizzi corrispondenti alla configurazione predefinita del servizio HTTP.

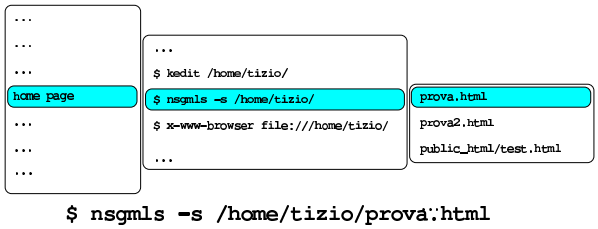
Tabella u29.4. Indirizzi URI della configurazione predefinita del servente HTTP.

Indirizzo	Descrizione
*http://**nodo**/a2/* *http://**nodo**/a2.pdf*	*a2*
*http://**nodo**/access/*	Statistiche di accesso al servizio.
*http://**nodo**/cgi/*	Esempi pronti di programmi CGI.
*http://**nodo**/cgi-bin/*	Corrisponde alla directory ‘`/usr/lib/cgi-bin/`’, di cui può essere visionato il contenuto. Contiene i programmi CGI dei servizi predisposti.
*http://**nodo**/cgi-bin/var_log*	Accesso ai registri del sistema (log), ammesso che siano disponibili i permessi di lettura nei file relativi.
*http://cgi-bin/**nodo**/info2www/*	Documentazione Info.
*http://**nodo**/dwww/*	Pagine di manuale e documentazione interna dei pacchetti applicativi installati.
*http://**nodo**/filter/*	Statistiche di accesso all’esterno, sotto il controllo di DansGuardian.
*http://**nodo**/mrtg/*	Statistiche del traffico di rete.
*http://**nodo**/php/*	Esempi di utilizzo del linguaggio PHP.

Esercitazioni con il linguaggio HTML

Per potersi esercitare nella realizzazione di file HTML aderenti allo standard è possibile utilizzare ‘**`nsgmls`**’, ma il menù grafico di NLNX viene in aiuto per una gestione ordinata del lavoro. Per questa funzionalità si parte da *home page*, come si vede nella figura successiva.

Figura u29.5. Funzioni del menù grafico di NLNX dedicate alla realizzazione e verifica di pagine HTML. In questo caso si viene guidati alla selezione di un file, per il suo controllo sintattico.



In questo caso, il menù grafico è sensibile ai file '`.html`' e '`.htm`', contenuti nella directory personale dell'utente e la directory '`~/public_html/`', ammesso che sia presente.

Interfacce di rete senza fili 121
Individuazione delle interfacce di rete 122
Connessione in una rete locale tradizionale 122
Connessione in una rete locale via radio 124
Connessione automatica in una rete non cifrata 126
Router per una rete locale 127
Router per una rete locale, attraverso un proxy 129
Indirizzi di rete da evitare assolutamente 132

Questo capitolo mostra alcune situazioni per cui NLNX è predisposto, per quanto riguarda il collegamento a una rete. Si osservi che NLNX è organizzato per funzionare correttamente in reti IPv4, anche senza la risoluzione dei nomi locali; pertanto, la lettura di questo capitolo e l'utilizzo relativo di NLNX presuppongono una conoscenza adeguata delle reti IPv4 (capitolo 32 e successivi).

Per quanto riguarda la gestione delle interfacce di rete senza filo, va tenuto in considerazione che NLNX le gestisce solo nella modalità di funzionamento *managed*, ovvero si richiede la presenza di almeno un punto di accesso. Se si vuole usare una modalità di tipo *ad-hoc*, lo script '**`nlnxrc`**' non può essere usato per gestire tali interfacce.

In un altro capitolo appare una tabella con le opzioni che si possono usare all'avvio, caratterizzate per avere il prefisso '**`n`**_' (tabella u17.2). Molte di quelle opzioni consentono di intervenire nella configurazione della rete, ma solo nel caso di un elaboratore con un'interfaccia di rete singola. L'uso di quelle opzioni prevale sulla configurazione memorizzata diversamente e anche sull'uso eventuale di un servente DHCP.

Interfacce di rete senza fili

Le interfacce di rete senza fili, se il kernel è in grado di gestirle, vengono configurate normalmente, nello stesso modo di quelle Ethernet, attraverso lo script '**`nlnxrc`**'. Tuttavia, può darsi che la propria interfaccia richieda l'uso di NDISwrapper per gestire un file binario fatto per MS-Windows. In tal caso, i file necessari a NDISwrapper vanno messi nella directory '`/etc/windows-drivers/wifi/`' e da lì vengono caricati automaticamente attraverso lo script '`/etc/init.d/nlnx.ndiswrapper`', chiamato a sua volta da '`/etc/init.d/nlnx.network`'.

A titolo di esempio, la propria interfaccia di rete potrebbe richiedere l'installazione di un *driver* attraverso il programma '**`setup.exe`**'. Questo programma, in realtà, incorpora alcuni file che vanno estratti, ma non disponendo di uno strumento adatto, occorre eseguirlo, attraverso WINE, il quale va usato da un terminale durante una sessione grafica di lavoro:

```
$ wine setup.exe [Invio]
```

Il programma mostra probabilmente delle finestre di dialogo per richiedere la conferma della licenza e del percorso in cui installare i file che servono a gestire l'interfaccia. In questo caso si presume che sia stato proposto (e confermato) il percorso '`C:\Program Files\NETGEAR\WG311v3\`'. Ovviamente, in un sistema GNU/Linux non esiste un tale percorso, ma nella gestione di WINE potrebbe corrispondere a '`~/.wine/drive_c/Program Files/NETGEAR/WG311v3/`'. Tra i vari file che possono trovarsi nella destinazione prevista, vanno scelti quelli con estensioni '`.inf`', '`.cat`' e '`.sys`'. Si suppone che siano stati ottenuti precisamente i file '`WG311v3.INF`', '`WG311v3.cat`', '`WG311v3.sys`' e '`WG311v3XP.sys`'.[1]

I file trovati vanno collocati nella directory '`/etc/windows-drivers/wifi/`', come già accennato. Se questi file sono quelli giusti, avviando manualmente il comando '`/etc/init.d/nlnx.`

`ndiswrapper`' si dovrebbe osservare che un'interfaccia '**`wlan`*`n`***' risulta attiva:

```
$ iwconfig [Invio]

  wlan1     IEEE 802.11g  ESSID:off/any
            Mode:Managed  Channel:0  Access Point: Not-Associated
            Bit Rate:1 Mb/s   Sensitivity=-200 dBm
            RTS thr=2346 B   Fragment thr=2346 B
            Power Management:off
            Link Quality:0  Signal level:0  Noise level:0
            Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
            Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Va però osservato che non si può produrre un DVD *live* o altra forma di NLNX per la distribuzione contenente tali file per MS-Windows, perché di norma si tratta di software proprietario (non libero). Pertanto, per assicurare che questo principio venga rispettato, durante la produzione del DVD, le directory '`/etc/windows-drivers/*/`' vengono svuotate automaticamente nella destinazione. Eventualmente, per forzare questa situazione, occorre modificare lo script '`/etc/init.d/nlnx.ndiswrapper`' in modo che carichi i file di gestione delle interfacce da un'altra collocazione. Sull'argomento si può leggere anche il capitolo 32.7.

Individuazione delle interfacce di rete

Le interfacce di rete Ethernet sono identificate da nomi del tipo '**`eth0`**', '**`eth1`**',... '**`eth`*`n`***', oppure '**`wlan0`**', '**`wlan1`**',... '**`wlan`*`n`***', nel caso di quelle per i collegamenti senza fili. Ma l'attribuzione del numero ***n*** alle interfacce non è controllabile. In pratica, può capitare che la prima e unica interfaccia di rete abbia il nome '**`eth7`**'. Quando si vogliono realizzare delle configurazioni abbastanza uniformi tra elaboratori che hanno una sola interfaccia di rete Ethernet, è improbabile che il nome attribuito all'interfaccia di rete sia lo stesso.

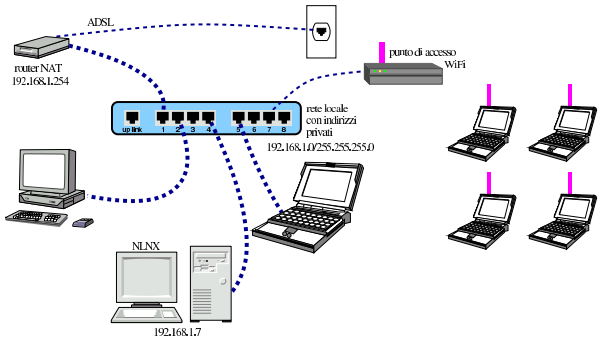
Con NLNX, quando si indica il nome di un'interfaccia Ethernet si può usare il nome effettivo ('**`eth0`**', '**`eth1`**',... '**`wlan0`**', '**`wlan1`**', ecc...), oppure si può fare riferimento a una variabile. Sono previste diverse variabili per questo scopo, da annotare esattamente così: '**`$ETH0`**', '**`$ETH1`**', '**`$ETH2`**', oppure '**`$WLAN0`**', '**`$WLAN1`**', '**`$WLAN2`**'. Questi nomi si riferiscono, rispettivamente, alla prima, alla seconda e alla terza interfaccia Ethernet, oppure alla prima, alla seconda e alla terza interfaccia WiFi, indipendentemente dal nome effettivo che gli viene attribuito automaticamente. Inoltre, è possibile usare la sigla '**`AUTO`**' per scegliere automaticamente la prima interfaccia, senza dover sapere se di tipo '**`eth`**...' o '**`wlan`**...'. Negli esempi dei capitoli, si intende utilizzare questa rappresentazione generica.

Connessione in una rete locale tradizionale

L'utilizzo di NLNX in una rete locale, che può disporre eventualmente di un router, si configura normalmente attraverso lo script '**`nlnxrc`**', senza bisogno di intervenire direttamente con '**`iwconfig`**', '**`ifconfig`**' e '**`route`**'.

La configurazione predefinita di NLNX prevede l'uso del protocollo DHCP, in modo da attribuire automaticamente l'indirizzo alla prima interfaccia di rete e anche l'instradamento necessario a uscire dalla rete locale.

Figura u30.2. Utilizzo di NLNX in una rete locale tipica.



La figura mostra una situazione pratica: l'elaboratore in cui è in funzione NLNX deve utilizzare l'indirizzo IPv4 192.168.1.7 (in quanto si trova nella rete 192.168.1.0/255.255.255.0) e può accedere all'esterno della rete locale attraverso un router NAT raggiungibile all'indirizzo 192.168.1.254. Per configurare NLNX attraverso '**`nlnxrc`**' si procede nel modo seguente:

```
# nlnxrc network config [Invio]
```

```
.------Internal network interface name--------.
| Please insert the real internal network     |
| interface name. To get auto configuration   |
| write "AUTO", to select the first Ethernet  |
| interface write "$ETH0", to select the      |
| first WLAN interface write "$WLAN0" and so  |
| on.                                         |
| .-----------------------------------------. |
| |$ETH0                                    | |
| '-----------------------------------------' |
|---------------------------------------------|
|          <  OK  >      <Cancel>             |
'---------------------------------------------'
```

`$ETH0` OK

Qui è stata specificata la variabile '**`$ETH0`**', per fare riferimento alla prima interfaccia di rete Ethernet che risulti essere disponibile. Ma trattandosi dell'unica interfaccia presente, avrebbe potuto essere inserita la voce generica '**`AUTO`**', ottenendo lo stesso risultato.

```
.------Internal network interface address---------.
| Please insert the internal network interface    |
| IPv4 address.                                   |
| If you want to use auto configuration with      |
| DHCP, please write the word "AUTO" instead of   |
| the IPv4 address.                               |
| .---------------------------------------------. |
| |                                             | |
| '---------------------------------------------' |
|-------------------------------------------------|
|            <  OK  >       <Cancel>             |
'-------------------------------------------------'
```

`192.168.1.7` OK

Come suggerisce il riquadro, volendo dichiarare esplicitamente di voler utilizzare il protocollo DHCP, al posto dell'indirizzo IPv4 si deve inserire la parola chiave '**`AUTO`**'. Ma in tal caso, le richieste successive non vengono fatte all'utente.

```
.--------Internal netmask----------.
| Please insert the internal IPv4  |
| network mask. Please note that   |
| the default netmask for the      |
| address class is 255.255.255.0:  |
| .------------------------------. |
| |255.255.255.0                 | |
| '------------------------------' |
|                                  |
|----------------------------------|
|       <  OK  >    <Cancel>       |
'----------------------------------'
```

`255.255.255.0` OK

```
.Internal network router address--.
| Please insert the internal      |
| network IPv4 router address, or |
| leave it blank if you don't     |
| have one:                       |
| .-----------------------------. |
| |                             | |
| '-----------------------------' |
|---------------------------------|
|     <  OK  >   <Cancel>         |
'---------------------------------'
```

192.168.1.254 OK

Si osservi che se la rete locale dovesse essere sprovvista di un router (una rete locale isolata), è importante evitare di indicare come indirizzo del router lo stesso indirizzo dell'interfaccia di rete locale, perché la concomitanza degli indirizzi fa presumere alla procedura prevista per NLNX che il nodo locale sia precisamente un router nei confronti della rete locale. Pertanto, **in mancanza di un router, il dato va lasciato in bianco**.

```
.-------------/etc/resolv.conf----------------.
| Do you need a default "/etc/resolv.conf"?   |
|                                             |
| Answer "No" if you want to keep it          |
| unchanged.                                  |
|---------------------------------------------|
|          < Yes >      < No  >              |
'---------------------------------------------'
```

YES

Un risultato equivalente avrebbe potuto essere ottenuto attraverso le opzioni di avvio seguenti:

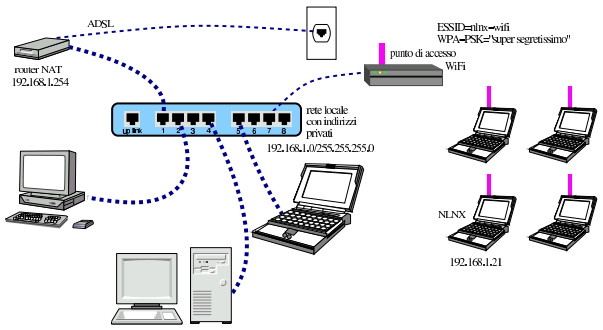
```
n_nic=$ETH0 n_ipv4=192.168.1.7
n_subnet_mask=255.255.255.0
n_router=192.168.1.254
```

Connessione in una rete locale via radio

«

NLNX può essere connesso alla rete locale anche con un'interfaccia WiFi, purché il kernel sia in grado di gestirla. Si procede in modo analogo a quanto già spiegato per la connessione tradizione. L'esempio successivo si riferisce all'elaboratore 192.168.1.21 che appare in figura.

Figura u30.8. Utilizzo di NLNX in una rete locale via radio.




```
# nlnxrc network config[Invio]
```

```
.------Internal network interface name--------.
| Please insert the real internal network     |
| interface name. To get auto configuration   |
| write "AUTO", to select the first Ethernet  |
| interface write "$ETH0", to select the      |
| first WLAN interface write "$WLAN0" and so  |
| on.                                         |
| .-----------------------------------------. |
| |$WLAN0                                   | |
| '-----------------------------------------' |
|---------------------------------------------|
|           <  OK  >      <Cancel>           |
'---------------------------------------------'
```


$WLAN0 OK

Come nel caso dell'interfaccia di rete Ethernet, qui si utilizza la variabile '**$WLAN0**' per fare riferimento genericamente alla prima interfaccia di rete via radio. Anche in questo caso sarebbe possibile inserire la parola chiave '**AUTO**', per fare riferimento genericamente alla prima interfaccia di rete disponibile, ma in questo modo non sarebbe possibile definire una parte della configurazione.[2]

```
.-INTERNAL interface ESSID---.
| Please insert the INTERNAL |
| interface ESSID string:    |
| .------------------------. |
| |nlnx-wifi               | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Trattandosi di un'interfaccia di rete senza fili, occorre specificare l'identificativo ESSID stabilito per comunicare con il punto di accesso. Si suppone che corrisponda proprio alla stringa «nlnx-wifi»:

nlnx-wifi OK

```
.-INTERNAL interface encryption selection---.
| Please select the encryption type:        |
| .---------------------------------------. |
| |   WPA-PSK  previous_value             | |
| |   NONE     clear text communication   | |
| |   WEP      WEP40/WEP104 encryption    | |
| |   WPA-PSK  WPA-PSK encryption         | |
| '---------------------------------------' |
|-------------------------------------------|
|         <  OK  >     <Cancel>             |
'-------------------------------------------'
```

Va quindi specificata il tipo di comunicazione. In questo caso si tratta di un collegamento cifrato secondo la modalità WPA con chiave segreta, di cui ogni nodo deve essere a conoscenza:

WPA-PSK OK

```
.INTERNAL interface WPA-PSK passphrase--.
| Insert the INTERNAL interface WPA-PSK |
| passphrase, writing an hexadecimal    |
| value or a string delimited with      |
| "...".                                |
| .-----------------------------------. |
| |                                   | |
| '-----------------------------------' |
|---------------------------------------|
|         <  OK  >     <Cancel>         |
'---------------------------------------'
```

Pertanto va specificata la chiave segreta, ovvero la parola d'ordine. In questo caso la parola d'ordine è racchiusa tra apici doppi, per precisare che si tratta di una stringa; diversamente, senza essere delimitata, si intenderebbe una sequenza esadecimale.

"supersegretissimo" OK

Si prosegue quindi con le informazioni consuete, tenendo conto che anche in questo caso ci si può avvalere del DHCP, indicando la parola chiave '**AUTO**', al posto dell'indirizzo IPv4:

```
.------Internal network interface address---------.
| Please insert the internal network interface    |
| IPv4 address.                                   |
| If you want to use auto configuration with      |
| DHCP, please write the word "AUTO" instead of   |
| the IPv4 address.                               |
| .---------------------------------------------. |
| |                                             | |
| '---------------------------------------------' |
|-------------------------------------------------|
|            <  OK  >       <Cancel>             |
'-------------------------------------------------'
```

192.168.1.21 OK

```
.--------Internal netmask----------.
| Please insert the internal IPv4  |
| network mask. Please note that   |
| the default netmask for the      |
| address class is 255.255.255.0:  |
| .------------------------------. |
| |255.255.255.0                 | |
| '------------------------------' |
|                                  |
|----------------------------------|
|       <  OK  >    <Cancel>       |
'----------------------------------'
```

255.255.255.0 OK

```
.Internal network router address--.
| Please insert the internal      |
| network IPv4 router address, or |
| leave it blank if you don't     |
| have one:                       |
| .------------------------------. |
| |                              | |
| '------------------------------' |
|----------------------------------|
|     <  OK  >   <Cancel>          |
'----------------------------------'
```

192.168.1.254 OK

```
.-------------/etc/resolv.conf----------------.
| Do you need a default "/etc/resolv.conf"?   |
|                                             |
| Answer "No" if you want to keep it          |
| unchanged.                                  |
|---------------------------------------------|
|           < Yes >      < No  >              |
'---------------------------------------------'
```

YES

Un risultato equivalente avrebbe potuto essere ottenuto attraverso le opzioni di avvio seguenti:

n_nic=$WLAN0 n_w_essid=nlnx-wifi n_w_encryption=WPA-PSK n_w_wpa_psk="supersegretissimo" n_ipv4=192.168.1.21 n_subnet_mask=255.255.255.0 n_router=192.168.1.254

Connessione automatica in una rete non cifrata

Per connettersi a una rete WiFi non cifrata, non è necessario conoscere l'identificativo ESSID. Pertanto, per la configurazione, questa informazione va lasciata in bianco:

```
.-INTERNAL interface ESSID---.
| Please insert the INTERNAL |
| interface ESSID string:    |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

OK

```
.-INTERNAL interface encryption selection---.
| Please select the encryption type:        |
| .--------------------------------------. |
| |   WPA-PSK  previous_value            | |
| |   NONE     clear text communication  | |
| |   WEP      WEP40/WEP104 encryption   | |
| |   WPA-PSK  WPA-PSK encryption        | |
| '--------------------------------------' |
|-------------------------------------------|
|         <  OK  >     <Cancel>            |
'-------------------------------------------'
```

NONE OK

...

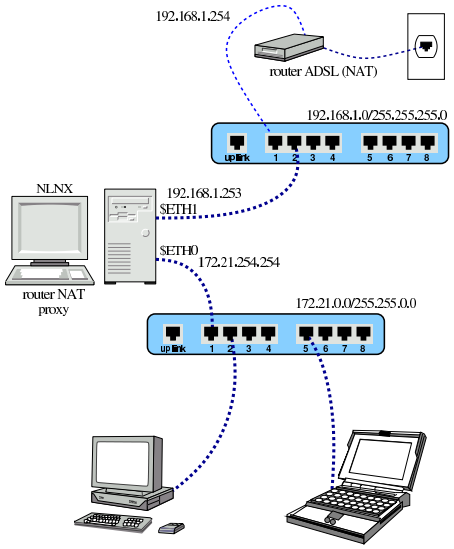
È da osservare che la configurazione predefinita che si ottiene dalla produzione di un nuovo DVD *live* (o equivalente) prevede proprio la selezione automatica dell'identificativo ESSID con una rete non cifrata.

Per questo tipo di configurazione, nella quale la comunicazione è in chiaro e l'identificativo ESSID non viene specificato, è sufficiente indicare la parola chiave '**AUTO**' al posto dell'interfaccia di rete.

Router per una rete locale

NLNX può essere usato anche per intervenire in qualità di router al servizio di una rete locale, per la connessione con una rete esterna. Si immagina una situazione simile a quella della figura successiva.

Figura u30.19. NLNX utilizzato come router.



Nella figura, il router si colloca tra due reti: 172.21.∗.∗ e 192.168.1.∗. Per la precisione, la rete 172.21.∗.∗ accede all'esterno attraverso la trasformazione degli indirizzi (NAT), perché si presume che gli instradamenti nella rete 192.168.1.∗ consentano di raggiungere l'esterno (Internet), ma non di accedere alla rete 172.21.∗.∗.

nlnxrc network config [*Invio*]

```
.------Internal network interface name--------.
| Please insert the real internal network     |
| interface name. To get auto configuration   |
| write "AUTO", to select the first Ethernet  |
| interface write "$ETH0", to select the      |
| first WLAN interface write "$WLAN0" and so  |
| on.                                         |
| .-----------------------------------------. |
| |$ETH0                                    | |
| '-----------------------------------------' |
|---------------------------------------------|
|           <  OK  >      <Cancel>            |
'---------------------------------------------'
```

$ETH0 OK

```
.Internal network interface address--.
| Please insert the internal network |
| interface IPv4 address:            |
| .--------------------------------. |
| |                                | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >    <Cancel>         |
'------------------------------------'
```

172.21.254.254 OK

```
.-----Internal netmask-------.
| Please insert the internal |
| IPv4 network mask:         |
| .------------------------. |
| |255.255.0.0             | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

255.255.0.0 OK

```
.Internal network router address--.
| Please insert the internal      |
| network IPv4 router address:    |
| .-----------------------------. |
| |                             | |
| '-----------------------------' |
|---------------------------------|
|     <  OK  >   <Cancel>         |
'---------------------------------'
```

172.21.254.254 OK

Dal momento che l'indirizzo del router per la rete interna coincide con l'indirizzo dell'interfaccia, la procedura intende che si debba specificare anche il collegamento con l'esterno:

```
.----External network interface name------.
| Please insert the real external network |
| interface name. To select the first     |
| Ethernet interface write "$ETH0", to    |
| select the first WLAN interface write   |
| "$WLAN0" and so on.                     |
| .-------------------------------------. |
| |$ETH1                                | |
| '-------------------------------------' |
|-----------------------------------------|
|         <  OK  >     <Cancel>          |
'-----------------------------------------'
```

Per funzionare correttamente, un router con le funzionalità che si richiedono qui deve avere due interfacce. Nel caso si trattasse di una sola interfaccia ci possono essere delle funzionalità che vengono a mancare, in ogni caso **si deve inserire sempre il nome reale dell'interfaccia, non un alias**.

$ETH1 OK

```
.External network interface address--.
| Please insert the external network |
| interface IPv4 address:            |
| .--------------------------------. |
| |                                | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >    <Cancel>         |
'------------------------------------'
```

192.168.1.253 OK

```
.-----External netmask-------.
| Please insert the external |
| IPv4 network mask:         |
| .------------------------. |
| |255.255.255.0           | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

255.255.255.0 OK


```
.External network router address--.
| Please insert the external      |
| network IPv4 router address:    |
| .-----------------------------. |
| |                             | |
| '-----------------------------' |
|---------------------------------|
|     <  OK  >   <Cancel>         |
'---------------------------------'
```

192.168.1.254 OK

```
.----Transparent proxy------.
| Is this router a          |
| transparent proxy?        |
|---------------------------|
|     < Yes >   < No  >     |
'---------------------------'
```

Potrebbe essere conveniente sfruttare il proxy imponendo il suo utilizzo da parte della rete locale:

YES

Il router che si ottiene si comporta anche come firewall, secondo una configurazione di massima che dovrebbe impedire alcuni tipi di accesso dall'esterno. Tuttavia, se esistono effettivamente dei problemi di sicurezza, la configurazione del firewall deve essere valutata personalmente da chi si incarica di realizzare una rete locale del genere; eventualmente è possibile modificare lo script '`/etc/init.d/nlnx.network`'. Le istruzioni che riguardano la configurazione in qualità di router iniziano a partire dalla porzione di codice evidenziata dal confronto tra l'indirizzo locale e l'indirizzo del router interno:

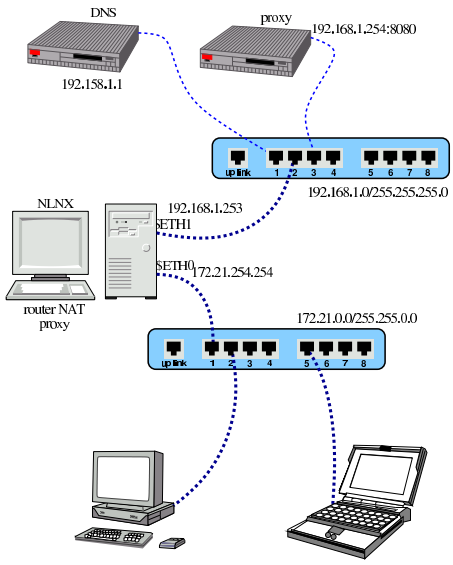
```
…
    elif    [ "$INTERNAL_IPV4" = "$INTERNAL_ROUTER" ]
    then
        ##
        ## This is the router for the local network.
        ##
…
```

L'esempio mostrato fa riferimento a indirizzi IPv4 privati sia dal lato interno, sia dal lato esterno del router. Con questo esempio si vuole individuare una situazione che potrebbe essere abbastanza comune: una rete locale gestita attraverso un router la cui configurazione non può essere cambiata, senza la disponibilità di indirizzi privati a sufficienza per le esigenze di tutte le reti. Con la soluzione proposta dall'esempio, si va a utilizzare un solo indirizzo nell'ambito della rete precedente, aggiungendo un altro router per un'altra rete locale che comunque sarebbe irraggiungibile nell'ambito di quella preesistente (per mancanza di instradamenti), pertanto si rende necessario il NAT nel nuovo router inserito.

Router per una rete locale, attraverso un proxy

Se non esiste altra possibilità di accedere alla rete esterna se non attraverso un proxy HTTP, è possibile tentare di organizzare la configurazione del router NLNX in modo che il proprio proxy faccia riferimento a quello disponibile. Questo è comunque condizionato alla disponibilità di un servizio di risoluzione dei nomi a dominio (DNS) accessibile. Si immagina una situazione simile a quella della figura successiva.

Figura u30.30. NLNX utilizzato come router che deve avvalersi di un proxy.



Come già visto in un esempio di un'altra sezione, nella figura, il router si colloca tra due reti: 172.21.∗.∗ e 192.168.1.∗.

```
# nlnxrc network config [Invio]
```

```
.-------Internal network interface name--------.
| Please insert the real internal network     |
| interface name. To get auto configuration   |
| write "AUTO", to select the first Ethernet  |
| interface write "$ETH0", to select the      |
| first WLAN interface write "$WLAN0" and so  |
| on.                                         |
| .-----------------------------------------. |
| |$ETH0                                    | |
| '-----------------------------------------' |
|---------------------------------------------|
|           <  OK  >      <Cancel>           |
'---------------------------------------------'
```

$ETH0 OK

```
.Internal network interface address--.
| Please insert the internal network |
| interface IPv4 address:            |
| .--------------------------------. |
| |                                | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >    <Cancel>         |
'------------------------------------'
```

172.21.254.254 OK

```
.-----Internal netmask-------.
| Please insert the internal |
| IPv4 network mask:         |
| .------------------------. |
| |255.255.0.0             | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

255.255.0.0 OK

```
.Internal network router address--.
| Please insert the internal      |
| network IPv4 router address:    |
| .-----------------------------. |
| |                             | |
| '-----------------------------' |
|---------------------------------|
|     <  OK  >   <Cancel>         |
'---------------------------------'
```


172.21.254.254 OK

Dal momento che l'indirizzo del router per la rete interna coincide con l'indirizzo dell'interfaccia, la procedura intende che si debba specificare anche il collegamento con l'esterno:

```
.----External network interface name------.
| Please insert the real external network |
| interface name. To select the first     |
| Ethernet interface write "$ETH0", to    |
| select the first WLAN interface write   |
| "$WLAN0" and so on.                     |
| .---------------------------------. |
| |$ETH1                            | |
| '---------------------------------' |
|-------------------------------------|
|         <  OK  >     <Cancel>       |
'-------------------------------------'
```

Per funzionare correttamente, un router con le funzionalità che si richiedono qui deve avere due interfacce. Nel caso si trattasse di una sola interfaccia ci possono essere delle funzionalità che vengono a mancare, in ogni caso **si deve inserire sempre il nome reale dell'interfaccia, non un alias**.

$ETH1 OK

```
.External network interface address--.
| Please insert the external network |
| interface IPv4 address:            |
| .--------------------------------. |
| |                                | |
| '--------------------------------' |
|------------------------------------|
|       <  OK  >    <Cancel>         |
'------------------------------------'
```

192.168.1.253 OK

```
.-----External netmask-------.
| Please insert the external |
| IPv4 network mask:         |
| .------------------------. |
| |255.255.255.0           | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

255.255.255.0 OK

```
.External network router address--.
| Please insert the external      |
| network IPv4 router address:    |
| .-----------------------------. |
| |                             | |
| '-----------------------------' |
|---------------------------------|
|     <  OK  >   <Cancel>         |
'---------------------------------'
```

In questo caso, non c'è alcun router esterno da poter raggiungere, pertanto si lascia il campo vuoto:

OK

```
.----Transparent proxy------.
| Is this router a          |
| transparent proxy?        |
|---------------------------|
|     < Yes >   < No  >     |
'---------------------------'
```

In questo caso, è necessario attivare la funzione di proxy imponendo il suo utilizzo da parte della rete locale:

YES

A questo punto, però, occorre fare delle modifiche manuali. Oltre a espandere la disponibilità di memoria di OOPS, è necessario che questo sia in grado di rinviare le richieste al proxy esterno. Si deve intervenire nei file '`/etc/oops/oops.cfg`' e '`/etc/oops/`

`oops.cfg.nlnx`' dell'elaboratore che offre questo servizio per la rete locale, aggiungendo le direttive seguenti:

```
peer    192.168.1.254   8080 0 {
        # my_auth       my_login:my_password;
        parent ;
        allow   dstdomain * ;
}
```

Per maggiori dettagli sulla configurazione di questa funzionalità di OOPS conviene consultare la sua documentazione originale.

La seconda modifica da apportare riguarda il servizio DNS: non potendo contare su un accesso alla rete esterna, il servente DNS di NLNX non serve a nulla ed è necessario modificare il file '`/etc/resolv.conf`' di tutti gli elaboratori della rete locale:

```
nameserver 192.168.1.1
```

Questa situazione potrebbe essere complicata ulteriormente se per l'accesso al proxy esterno o al servente DNS è necessario utilizzare un router. In tal caso si comprende che è sufficiente specificare l'indirizzo di tale router esterno, senza lasciare il campo in bianco come è stato fatto negli esempi mostrati in questa sezione.

Indirizzi di rete da evitare assolutamente

Onde evitare inutili perdite di tempo, è bene rammentare che nelle reti con indirizzi privati è necessario evitare alcuni indirizzi di rete, che apparentemente sono innocui. La tabella seguente riepiloga le situazioni più comuni, tenendo conto delle maschere di rete predefinite.

Indirizzo	Maschera	Motivazione
172.16.0.0	255.255.0.0	L'indirizzo di rete della sottorete è identico all'indirizzo della rete complessiva.
172.31.0.0	255.255.0.0	L'indirizzo broadcast della sottorete è identico all'indirizzo broadcast della rete complessiva.
192.168.0.0	255.255.255.0	L'indirizzo di rete della sottorete è identico all'indirizzo della rete complessiva.
192.168.255.0	255.255.255.0	L'indirizzo broadcast della sottorete è identico all'indirizzo broadcast della rete complessiva.

[1] Se l'operazione di estrazione con l'aiuto di WINE fallisce, occorre cercare aiuto in rete, con i dati dell'interfaccia.

[2] Scegliendo la voce '**`AUTO`**', in presenza di un'interfaccia di rete senza fili e di una tradizionale, ha la precedenza quella senza fili, ma la parte di configurazione che non può essere stabilita espressamente prevede una comunicazione in chiaro con qualunque identificativo ESSID.

VNC con «nlnxrc»

Organizzazione e funzionamento 133
Utilizzo di un elaboratore remoto con un pubblico passivo ... 134
Utilizzo di un elaboratore locale con un pubblico passivo 135
Utilizzo di un elaboratore remoto con un pubblico attivo 135
Utilizzo di un elaboratore locale, con un pubblico attivo 136
Utilizzo senza parola d'ordine 136
Utilizzo di VNC attraverso un tunnel SSH 137

NLNX prevede un utilizzo semplificato di VNC (sezione 28.13), attraverso l'uso dello script '**`nlnxrc`**'. Il sistema ha lo scopo di facilitare sia la realizzazione di lezioni in video-conferenza, sia un accesso remoto personale.

Per quanto riguarda la gestione della videoconferenza, il meccanismo proposto dallo script '**`nlnxrc`**' è molto semplice e non prevede sistemi di sicurezza ferrei, per impedire che qualcuno si intrometta nella comunicazione.

Tabella u16.9. Script '**`nlnxrc`**': controllo di VNC.

Comando	Descrizione
`nlnxrc vncs`	Si avvia preferibilmente da una console per attivare un servente VNC, definendo una parola d'ordine.
`nlnxrc vncsc`	Attiva un servente VNC, definendo una parola d'ordine, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc vncss`	Si avvia preferibilmente da una console per attivare un servente VNC condivisibile, definendo una parola d'ordine.
`nlnxrc vncssc`	Attiva un servente VNC condivisibile, definendo una parola d'ordine, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc vncv` ***nodo*** `nlnxrc vncv-ssh` ***nodo***	Consente di visualizzare il servente VNC in funzione presso il nodo indicato. La seconda delle due forme di utilizzo, implica la creazione di un tunnel SSH per garantire un collegamento cifrato.
`nlnxrc vncc` ***nodo*** `nlnxrc vncc-ssh` ***nodo***	Consente di interagire con il servente VNC in funzione presso il nodo indicato. La seconda delle due forme di utilizzo, implica la creazione di un tunnel SSH per garantire un collegamento cifrato.
`nlnxrc sharedx`	Attiva un servente VNC condivisibile, utilizzando una parola d'ordine predefinita, assieme al cliente VNC necessario a interagire con questo.
`nlnxrc viewremotex` ***nodo***	Consente di visualizzare il servente VNC in funzione presso il nodo indicato, utilizzando la parola d'ordine predefinita. In pratica si usa per collegarsi a un servente VNC avviato con il comando '**`nlnxrc sharedx`**'.

Organizzazione e funzionamento

Le sigle dei vari comandi di '**`nlnxrc`**' hanno lo scopo di sintetizzare il senso degli stessi, come descritto nella tabella successiva.

Tabella u31.1. Sigle mnemoniche utilizzate.

Nome	Descrizione
`vncs`	*VNC server*
`vncsc`	*VNC server with client*
`vncss`	*VNC shared server*
`vncssc`	*VNC shared server with client*
`vncc`	*VNC client*
`vncv`	*VNC viewer*

VNC richiede di definire una parola d'ordine per autorizzare il collegamento tra servente e cliente. Con lo script '**`nlnxrc`**', questa parola d'ordine viene annotata nel file '`~/.vnc/passwd`' e può essere riutilizzata; eventualmente, alcuni comandi fanno uso di una parola d'ordine predefinita, all'interno di '**`vncrc`**' stesso.

Tutto il meccanismo previsto da '**`nlnxrc`**' è organizzato in modo tale da far funzionare il servente VNC sulla stazione grafica '**`:1`**', pertanto questa informazione non viene mai impartita, ma di conseguenza, in un certo elaboratore, è possibile avviare un solo servente VNC per volta.

I comandi che avviano un servente VNC richiedono di specificare espressamente la geometria dello schermo:

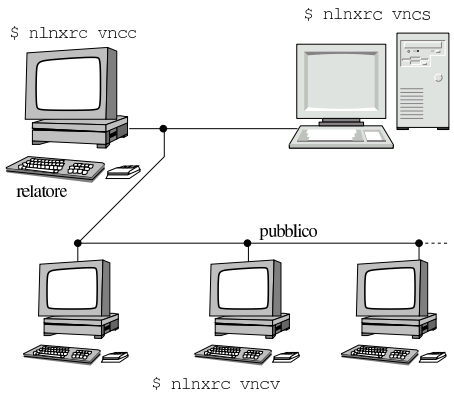
```
.-----------------geometry-------------------.
| Select geometry:                           |
| .---^(-)---------------------------------. |
| |   1160x928   5:4                      | |
| |   1184x888   4:3                      | |
| |   1320x792   5:3                      | |
| |   1120x896   5:4                      | |
| |   1152x864   4:3                      | |
| |   1280x768   5:3                      | |
| |   1120x840   4:3                      | |
| |   1080x864   5:4                      | |
| |   1240x744   5:3                      | |
| |   1088x816   4:3                      | |
| |   1040x832   5:4                      | |
| |   1200x720   5:3                      | |
| |   1056x792   4:3                      | |
| |   1160x696   5:3                      | |
| |   1000x800   5:4                      | |
| |   1024x768   4:3 usual resolution     | |
| |   1120x672   5:3                      | |
| `---v(+)---------------------------------' |
|--------------------------------------------|
|          <  OK  >      <Cancel>            |
`--------------------------------------------'
```

Naturalmente, se si vuole accedere al servente VNC attraverso una finestra, conviene utilizzare una geometria leggermente inferiore a quella dello schermo che si ha effettivamente a disposizione. Per esempio, se si utilizza il sistema grafico a una risoluzione di 1024×768, può essere conveniente avviare il servente VNC a 960×720.

Utilizzo di un elaboratore remoto con un pubblico passivo

Si ipotizza la situazione in cui, per qualche ragione, si vuole utilizzare X presso un elaboratore remoto, offrendo ad altri la possibilità di visualizzare ciò che succede. Per fare questo occorre avviare presso l'elaboratore remoto il comando '**`nlnxrc vncs`**', mentre localmente si può utilizzare il comando '**`nlnxrc vncc`**' per poterlo controllare e '**`nlnxrc vncv`**' per la sola visualizzazione.

Figura u31.3. Utilizzo di un servente X presso un elaboratore remoto, con un pubblico passivo.



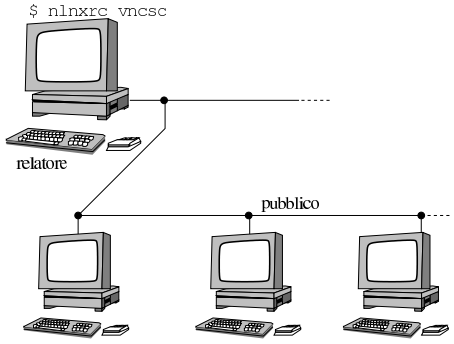
All'avvio del servente grafico viene richiesto di inserire una parola d'ordine, o di riutilizzare quella che si trova nel file '`~/.vnc/passwd`'. La parola d'ordine è l'unico mezzo reale per controllare l'accesso al servente grafico e deve essere fornita anche a chi visualizza o partecipa al controllo.

Il meccanismo proposto, vale in quanto ci si attende un comportamento corretto da parte del pubblico. Infatti, se invece di '**`nlnxrc vncv`**' si utilizza il comando '**`nlnxrc vncc`**', si ruba la sessione di controllo del servente grafico a chi invece ha il compito di svolgere la relazione.

Utilizzo di un elaboratore locale con un pubblico passivo

In questo caso si vuole riprodurre una situazione equivalente a quella della sezione precedente, dove però il servente grafico si trova presso lo stesso elaboratore locale del relatore. In tal caso, l'avvio del servente grafico si ottiene con il comando '**`nlnxrc vncsc`**', offrendo agli altri la possibilità di visualizzare con il comando '**`nlnxrc vncv`**'. Per il resto valgono le considerazioni già fatte nella sezione precedente.

Figura u31.4. Utilizzo di un servente X presso l'elaboratore locale, con un pubblico passivo.



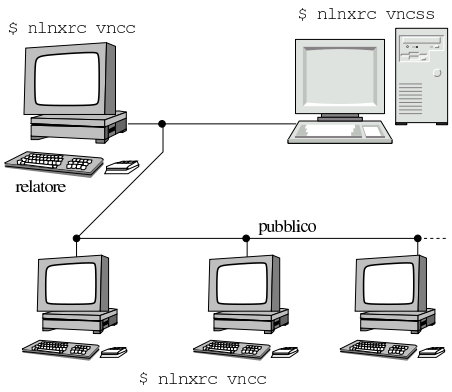
Come già annotato nella sezione precedente, il meccanismo proposto, vale in quanto ci si attende un comportamento corretto da parte del pubblico. Infatti, se invece di '**`nlnxrc vncv`**' si utilizza il comando '**`nlnxrc vncc`**', si ruba la sessione di controllo del servente grafico a chi invece ha il compito di svolgere la relazione.

Utilizzo di un elaboratore remoto con un pubblico attivo

Si ipotizza la situazione in cui, per qualche ragione, si vuole utilizzare X presso un elaboratore remoto, offrendo a tutti la possibilità di

interagirvi. Per fare questo occorre avviare presso l'elaboratore remoto il comando '**`nlnxrc vncss`**', mentre localmente si utilizzare il comando '**`nlnxrc vncc`**'.

Figura u31.5. Utilizzo di un servente X presso un elaboratore remoto con un pubblico attivo.

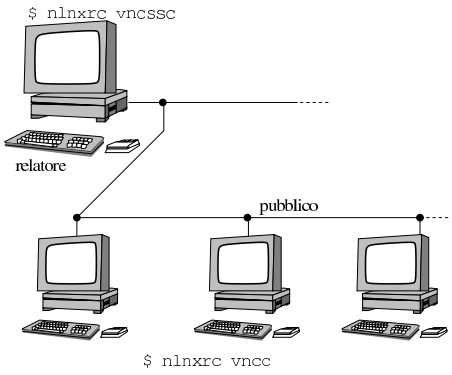


Rispetto alla descrizione delle sezioni precedenti, cambia il fatto che ogni utente partecipa al controllo del servente remoto.

Utilizzo di un elaboratore locale, con un pubblico attivo

In questo caso si vuole riprodurre una situazione equivalente a quella della sezione precedente, dove però il servente grafico che si vuole condividere, si trova presso lo stesso elaboratore che partecipa al suo controllo. L'avvio del servente grafico si ottiene con il comando '**`nlnxrc vncssc`**', mentre il controllo partecipativo del pubblico avviene sempre con il comando '**`nlnxrc vncc`**'.

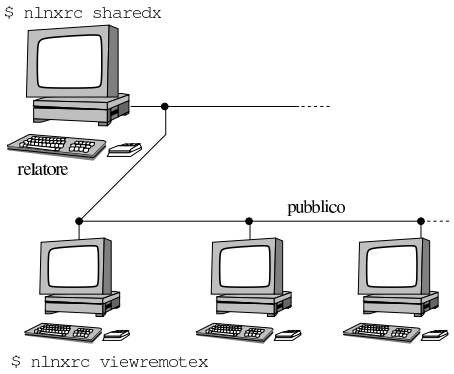
Figura u31.6. Utilizzo di un servente X presso l'elaboratore locale.



Utilizzo senza parola d'ordine

Come già accennato, c'è la possibilità di usare VNC utilizzando una parola d'ordine predefinita, che così può semplificare il lavoro, almeno in una rete locale che dia garanzie sufficienti di isolamento. Per questo è possibile avviare un servente grafico locale, attraverso i comandi '**`nlnxrc sharedx`**' e lasciare che il pubblico veda con il comando '**`nlnxrc viewremotex`**'.

Figura u31.7. Un relatore mostra qualcosa al pubblico.

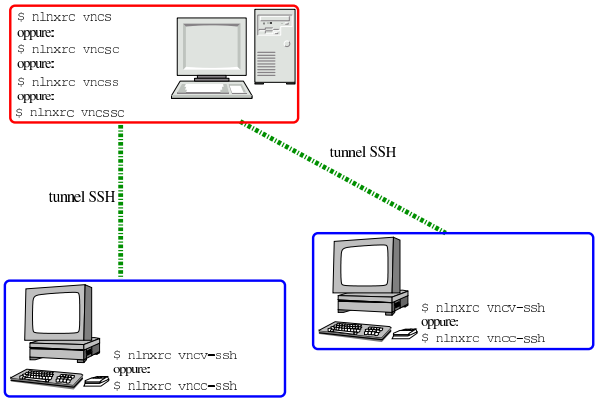


Utilizzo di VNC attraverso un tunnel SSH

I comandi '**`nlnxrc vncv`**' e '**`nlnxrc vncc`**', hanno delle alternative, costituite rispettivamente da: '**`nlnxrc vncv-ssh`**' e '**`nlnxrc vncc-ssh`**'. Nel secondo caso, il collegamento verso l'elaboratore remoto avviene tramite un tunnel cifrato SSH (Secure Shell).

Si osservi che dal lato del servente VNC non si deve fare nulla di diverso e si utilizzano i comandi già descritti: '**`nlnxrc vncs`**', '**`nlnxrc vncsc`**', '**`nlnxrc vncss`**' o '**`nlnxrc vncssc`**'.

Figura u31.8. Accesso a un servente X presso un elaboratore remoto, utilizzando un tunnel SSH.



Il comportamento dei comandi '**`vncv-ssh`**' e '**`vncc-ssh`**', è lo stesso di quelli a cui si abbinano (senza l'estensione '**`-ssh`**'), con la differenza che per la creazione del tunnel serve l'indicazione del nominativo utente per conto del quale deve essere realizzato, con la richiesta eventuale di una parola d'ordine):

```
.----SSH tunnel for VNC------.
| Please enter the user name |
| to access the remote VNC   |
| server at                  |
| 172.21.254.254:1.          |
| .------------------------. |
| |tizio                   | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'

.--------SSH tunnel---------.
| You are going to enter    |
| the password to access    |
| 172.21.254.254 with SSH.  |
'---------------------------'

Password:
```

Evidentemente, se il tunnel non può essere instaurato, non può avvenire la connessione.

Al termine del collegamento, il tunnel viene eliminato; tuttavia, quando si tenta di avviare un altro collegamento, se esiste già un tunnel dello stesso tipo richiesto, si può tentare di riutilizzarlo:

```
.--------------------------------SSH tunnel--------------------------------.
| There is already a tunnel with port 5901:                                |
|                                                                          |
| 19056 ?    Ss   0:00 ssh -N -f -L 5901:localhost:5901 tizio@172.21.254.254 |
|                                                                          |
| What should I do?                                                        |
| .----------------------------------------------------------------------. |
| |                  reuse    try to reuse the old tunnel                | |
| |                  kill     try to kill the SSH process                | |
| |                  abandon  don't do anything                          | |
| '----------------------------------------------------------------------' |
|--------------------------------------------------------------------------|
|                      < OK >            <Cancel>                          |
'--------------------------------------------------------------------------'
```


Creazione, modifica e cancellazione delle utenze, in modo interattivo 140
Creazione, modifica e cancellazione delle utenze, in modo non interattivo 143
NIS e NFS per utilizzare altre utenze 144
Controllo dello spazio utilizzato 145
Controllo del numero di pagine stampabili e dell'origine delle stampe 147
Utenze speciali per l'amministrazione 148
`print-filter` 147

In generale, la gestione delle utenze da parte di NLNX è quella tradizionale dei sistemi Unix, con delle semplificazioni che facilitano la condivisione delle utenze tramite l'uso di un servente NFS, NIS e Samba.

Tabella u16.7. Script '**`nlnxrc`**': gestione delle utenze.

Comando	Descrizione
`nlnxrc user add` [*utente* ↩ ↪ [*home_dir* ↩ ↪ [*descrizione* ↩ ↪ [*parola_d'ordine*]]]] `nlnxrc user del` [*utente*]	Aggiunge o elimina un'utenza, secondo la procedura completa prevista da NLNX.
`nlnxrc user passwd` [*utente* ↩ ↪ [*parola_d'ordine*]]	Cambia la parola d'ordine di un utente, secondo la procedura completa prevista da NLNX.
`nlnxrc user info` `nlnxrc home info`	Consente di avere informazioni sugli utenti comuni, partendo, rispettivamente, da un elenco in ordine alfabetico del nominativo utente, oppure della directory personale.
`nlnxrc quota set`	Se è attiva la gestione delle quote di utilizzo della memoria di massa, consente di stabilire il limite di spazio per le utenze.
`nlnxrc quota report`	Se è attiva la gestione delle quote di utilizzo della memoria di massa, visualizza la situazione di tutti gli utenti, ordinando l'elenco partendo da quelli che stanno utilizzando più spazio.
`nlnxrc nis-server-users edit`	Per motivi di sicurezza, indica quali utenti comuni possono accedere direttamente all'elaboratore che offre il servizio NIS.
`nlnxrc machine add` `nlnxrc machine del` [*nome*]	Aggiunge o elimina un'utenza speciale, associata a un elaboratore MS-Windows, per la gestione degli accessi da tale sistema operativo, attraverso Samba.
`nlnxrc admin add` [*nominativo* ↩ ↪ [*descrizione* ↩ ↪ [*parola_d'ordine*]]] `nlnxrc admin del` [*nominativo*]	Aggiunge o elimina un'utenza amministrativa. Per la precisione, si ottiene un amministratore con nome ***nominativo*** dal lato NLNX e uno con nome **`win.`*nominativo*** dal lato MS-Windows.
`nlnxrc admin passwd` ↩ ↪ [*nominativo* [*parola_d'ordine*]]	Cambia la parola d'ordine di un amministratore, secondo la procedura completa prevista da NLNX.

Tabella u16.3. Script '**`nlnxrc`**': configurazione del servizio NIS.

Comando	Descrizione
`nlnxrc nis-server config` `nlnxrc nis-server unconf`	Attiva o disattiva il funzionamento in qualità di servente NIS.

Comando	Descrizione
`nlnxrc nis-server-users edit`	Per motivi di sicurezza, indica quali utenti comuni possono accedere direttamente all'elaboratore che offre il servizio NIS.
`nlnxrc nis stop`	Disattiva le funzioni NIS (sia come servente, sia come cliente).

Creazione, modifica e cancellazione delle utenze, in modo interattivo

«

È necessario gestire la creazione e l'eliminazione delle utenze attraverso lo script '**`nlnxrc`**', per garantire che l'organizzazione di NLNX sia preservata. In particolare, le directory personali degli utenti sono raggruppate per categoria, a partire dalla directory '`/home/`'. Per esempio, la directory '`/home/LUCE/chiara/`' potrebbe essere la directory personale dell'utente '**`chiara`**' che in qualche modo viene considerato parte della categoria '**`LUCE`**'.

Si osservi che NLNX è organizzato in modo da gestire i gruppi privati, pertanto la classificazione descritta degli utenti serve solo a raggruppare le directory personali per facilitare il lavoro di amministrazione.

Un altro elemento distintivo delle utenze di NLNX consiste nei permessi iniziali che vengono dati alle directory personali degli utenti. Queste appartengono in realtà a '**`root`**' e al gruppo privato dell'utente a cui è affidata la directory, pertanto vengono dati tutti i permessi di accesso, sia al proprietario, sia al gruppo, ma agli altri utenti rimane solo il permesso di accesso, senza la possibilità di leggere il contenuto. Nell'ipotesi dell'utente '**`chiara`**' già descritta, si possono vedere i permessi seguenti:

```
# ls -l /home/LUCE [Invio]

…
drwxrwx--x  2 root    chiara    4096 2012-06-14 09:50 chiara
…
```

Questo accorgimento serve a garantire che gli utenti non possano cambiare i permessi della propria directory personale, anche se i permessi dei contenuti rimangono sotto il loro controllo.

A partire dalla directory '`/home/.samba/profiles/`' vengono anche create, automaticamente, delle sottodirectory con gli stessi nomi degli utenti e i permessi per consentire loro l'accesso. Tali directory servono a Samba, per consentire la gestione dei profili personali quando si accede attraverso sistemi MS-Windows. In tal caso, con l'utenza '**`chiara`**', si ottiene la directory '`/home/.samba/profiles/chiara/`'.

Per inserire una nuova utenza, si comincia con il comando seguente:

```
# nlnxrc user add [Invio]
```

```
.---------------------Add a new user-----------------------.
| [en] Please insert the new user name (only lower case    |
| letters and numbers, minimum 8 and max 15 characters,    |
| the first character must be a letter).                   |
|                                                          |
| [it] Inserire il nominativo utente, composto             |
| preferibilmente da cognome e nome attaccati, ed          |
| eventualmente l'anno di nascita, per esempio             |
| "rossimario1990" (si possono usare solo lettere          |
| minuscole e cifre numeriche per un minimo di 8 e fino a  |
| un massimo di 15 caratteri, ma il primo carattere deve   |
| essere una lettera).                                     |
| |........9012345|                                      |
| .-------------------------------------------------------. |
| |                                                       | |
| '-------------------------------------------------------' |
|----------------------------------------------------------|
|               <  OK  >           <Cancel>               |
'----------------------------------------------------------'
```

Il nominativo utente da inserire può essere lungo al massimo 15 caratteri, composti da lettere minuscole dell'alfabeto inglese e da

cifre numeriche, senza spazi. Supponendo di voler creare l'utente '**`rossimario`**', si procede così:

`rossimario` OK

```
.--------------------Classify user-----------------------.
| [en] Please select a hierarchy name for the user: it   |
| will be used as an intermediate directory after        |
| "/home/".                                              |
|                                                        |
| [it] Inserire la classificazione dell'utente: viene    |
| usata come directory intermedia dopo "/home/".         |
| .--------------------------------------------------. |
| |             1AI1213   /home/1AI1213              | |
| |             1AM1213   /home/1AM1213              | |
| |             1AS1213   /home/1AS1213              | |
| |             1BI1213   /home/1BI1213              | |
| |             1BM1213   /home/1BM1213              | |
| |             1BS1213   /home/1BS1213              | |
| |             1CI1213   /home/1CI1213              | |
| |             1CM1213   /home/1CM1213              | |
| |             1DI1213   /home/1DI1213              | |
| |             --------  -------------------------  | |
| |             new       add a new one              | |
| '-------------v(+)-----------------------------------' |
|                                                        |
|                                                        |
|--------------------------------------------------------|
|              <  OK  >        <Cancel>                 |
'--------------------------------------------------------'
```

Viene richiesto di specificare la classificazione da dare all'utenza, proponendo un elenco con le directory che sembrano essere già state usate con questo scopo. I nomi usati per la classificazione possono contenere al massimo otto caratteri, a scelta tra lettere maiuscole e cifre numeriche.. Si suppone di voler attribuire all'utente la classificazione '**`5A1213`**' che non risulta ancora prevista, pertanto si seleziona la voce '**`new`**'.

```
.---------------------Classify user-------------------------.
| [en] Please insert a hierarchy name for the user: it     |
| will be used as an intermediate directory after          |
| "/home/". Please insert only upper case letters and      |
| digits (A-Z, 0-9; min 4, max 8 characters).              |
|                                                          |
| [it] Inserire la classificazione dell'utente: viene      |
| usata come directory intermedia dopo "/home/".           |
| Inserire solo lettere maiuscole e cifre numeriche (A-Z,  |
| 0-9; min 4, max 8 caratteri).                            |
| |....5678|                                               |
| .-------------------------------------------------------. |
| |2012                                                   | |
| '-------------------------------------------------------' |
|                                                          |
|----------------------------------------------------------|
|                <  OK  >           <Cancel>               |
'----------------------------------------------------------'
```

Viene proposto l'anno attuale, ma viene cambiato secondo quanto già previsto:

[*Canc*][*Canc*][*Canc*][*Canc*]

`5A1213` OK

```
.--------------------Full user name----------------------.
| [en] Please insert the user full name and maybe some   |
| more data to identify it.                              |
|                                                        |
| [it] Inserire la descrizione dettagliata dell'utente,  |
| per poterlo identificare con precisione.               |
| .--------------------------------------------------. |
| |                                                  | |
| '--------------------------------------------------' |
|--------------------------------------------------------|
|              <  OK  >        <Cancel>                 |
'--------------------------------------------------------'
```

Viene richiesto di specificare il nome completo; in questo caso si indica anche la classe e l'anno scolastico. Si devono evitare la virgola e i due punti, che comunque verrebbero omessi automaticamente:

`studente Rossi Mario 5A 2012/2013` [*Invio*]

```
Adding user ‘rossimario’...
Adding new group ‘rossimario’ (1000).
Adding new user ‘rossimario’ (1000) with group ‘rossimario’.
Creating home directory ‘/home/5A1213/rossimario’.
Copying files from ‘/etc/skel’
```

Al termine viene richiesto di inserire per due volte la parola d’ordine, cosa che deve fare direttamente la persona interessata:

digitazione_all’oscuro OK

```
.------------------------New password----------------------.
| [en] Please insert the new password for user             |
| "rossimario" with home directory                         |
| "/home/5A1213/rossimario". Please insert at least 7      |
| characters.                                              |
|                                                          |
| [it] Inserire la nuova parola d’ordine per l’utente      |
| "rossimario" che dispone della directory personale       |
| "/home/5A1213/rossimario".                               |
| Si prega di inserire almeno 7 caratteri.                 |
|  <-MIN->                                                 |
| .--------------------------------------------------. |
| |************                                      | |
| ‘--------------------------------------------------’ |
|----------------------------------------------------------|
|               <  OK  >          <Cancel>                |
‘----------------------------------------------------------’
```

L’inserimento corrisponde alla visualizzazione di una serie di asterischi.

digitazione_all’oscuro OK

```
.-----------------------New password-------------------------.
| [en] Please insert again the new password for user         |
| "rossimario" with home directory "/home/5A1213/rossimario".|
|                                                            |
| [it] Inserire nuovamente la parola d’ordine per l’utente   |
| "rossimario" che dispone della directory personale         |
| "/home/5A1213/rossimario".                                 |
|  <-MIN->                                                   |
| .--------------------------------------------------. |
| |************                                      | |
| ‘--------------------------------------------------’ |
|------------------------------------------------------------|
|                <  OK  >         <Cancel>                   |
‘------------------------------------------------------------’
```

Dopo l’inserimento, per due volte, della parola d’ordine, se tutto è stato fatto senza errori, soprattutto se la parola d’ordine è stata inserita correttamente si conclude l’operazione ed eventualmente si può procedere con l’inserimento di un utente successivo. In questo caso si vuole smettere:

CANCEL

Si osservi che se si tenta di inserire un nominativo-utente più lungo del consentito, questo viene tagliato al quindicesimo carattere, senza mostrare errori.

L’organizzazione di NLNX richiede che anche il cambiamento della parola d’ordine avvenga attraverso un comando di ‘**nlnxrc**’, avviato di norma dall’amministratore. Ciò dipende dal fatto che NLNX è pensato principalmente per l’utilizzo in rete, con la condivisione delle utenze attraverso il NIS e Samba. Questa limitazione consente di mantenere valido l’abbinamento tra utenze Unix, NIS e Samba, ma richiede di intervenire presso il servente; l’esempio seguente mostra l’amministratore che cambia la parola d’ordine di un utente comune:

```
# nlnxrc user passwd [Invio]
```



```
.---------------Change password------------------.
| [en] Please insert the user name who have to   |
| change the password.                           |
|                                                |
| [it] Inserire il nominativo utente per il      |
| quale si deve cambiare la parola d’ordine.     |
| .--------------------------------------. |
| |                                      | |
| ‘--------------------------------------’ |
|------------------------------------------------|
|           <  OK  >      <Cancel>              |
‘------------------------------------------------’
```

rossimario OK

```
.-------------------------New password--------------------------.
| [en] Please insert the new password for user "rossimario" with    |
| home directory "/home/5A1213/rossimario". Please insert at least 7 |
| characters.                                                       |
|                                                                   |
| [it] Inserire la nuova parola d’ordine per l’utente "rossimario"  |
| che dispone della directory personale "/home/5A1213/rossimario".  |
| Si prega di inserire almeno 7 caratteri.                          |
|  <-MIN->                                                          |
| .-------------------------------------------------------------. |
| |************                                                 | |
| ‘-------------------------------------------------------------’ |
|-------------------------------------------------------------------|
|                <  OK  >         <Cancel>                          |
‘-------------------------------------------------------------------’
```

digitazione_all’oscuro OK

```
.-----------------------New password-------------------------.
| [en] Please insert again the new password for user         |
| "rossimario" with home directory "/home/5A1213/rossimario".|
|                                                            |
| [it] Inserire nuovamente la parola d’ordine per l’utente   |
| "rossimario" che dispone della directory personale         |
| "/home/5A1213/rossimario".                                 |
|  <-MIN->                                                   |
| .--------------------------------------------------. |
| |************                                      | |
| ‘--------------------------------------------------’ |
|------------------------------------------------------------|
|                <  OK  >         <Cancel>                   |
‘------------------------------------------------------------’
```

Per eliminare un’utenza si procede in modo simile all’inserimento:

```
# nlnxrc user del [Invio]
```

```
.----Delete an old user------.
| Please insert the user     |
| name to be removed.        |
| .------------------------. |
| |                        | |
| ‘------------------------’ |
|----------------------------|
|   <  OK  >  <Cancel>       |
‘----------------------------’
```

rossimario OK

Creazione, modifica e cancellazione delle utenze, in modo non interattivo

Nella sezione precedente è illustrato il meccanismo di creazione, modifica e cancellazione delle utenze, in modo interattivo, dove lo script ‘**nlnxrc**’ richiede mano a mano le informazioni necessarie all’operatore. Tuttavia, i comandi ‘**nlnxrc user add**’, ‘**nlnxrc user passwd**’ e ‘**nlnxrc user del**’, accettano degli argomenti, così da consentire di svolgere le loro operazioni senza bisogno di un intervento umano; in pratica, in questo modo, è possibile inserire tali comandi all’interno di altri script.

Gli argomenti previsti dai comandi in questione, devono essere forniti in modo completo, altrimenti, le informazioni mancanti vengono richieste in modo interattivo. A titolo di esempio viene mostrato in che modo creare e modificare ed eliminare l’utenza «rossimario», usata nella sezione precedente.

```
# nlnxrc user add rossimario /home/5A1213/rossimario ↩
↪                "studente Rossi Mario 5A 2012/2013" ↩
↪                segretissimo1 [Invio]

# nlnxrc user passwd rossimario supersegretissimo2 [Invio]

# nlnxrc user del rossimario [Invio]
```

NIS e NFS per utilizzare altre utenze

NLNX è predisposto inizialmente con pochi utenti: l'amministratore e alcuni utenti comuni. Tuttavia, è disponibile un sistema NIS per la connessione a un servente NIS, dal quale ottenere le informazioni su altre utenze (precisamente per i file '`/etc/passwd`', '`/etc/shadow`' e '`/etc/group`'). In pratica, dal momento che l'utilizzo con un file system in sola lettura (come il DVD *live*) comporta delle limitazioni, è prevista la possibilità di acquisire queste utenze dall'esterno, innestando le directory personali di queste utenze a partire dalla directory '`/home/`', attraverso il protocollo NFS.

Per l'utilizzo di questa funzionalità, è necessario:

- un servente NIS che metta a disposizione le informazioni dei file '`/etc/passwd`', '`/etc/shadow`' e '`/etc/group`', tenendo conto che se la condivisione del file '`/etc/shadow`' non funziona bene, potrebbe essere necessario disabilitare presso quel servente l'uso delle parole d'ordine oscurate (*shadow password*);
- un servente NFS che metta a disposizione la directory '`/home/`', la quale deve contenere le directory personali degli utenti gestiti tramite il servente NIS (se il servente NFS offre la condivisione di un'altra directory, questa non potrebbe essere creata in un DVD *live*, pertanto, in tal caso non vi si potrebbe accedere).

Se sono rispettati questi requisiti, si può attivare la gestione di queste utenze remote attraverso il comando '**`nlnxrc nis-home start`**':

```
# nlnxrc nis-home start [Invio]
```

```
.--------NIS domain--------.
|                           |
| Please insert the NIS     |
| domain:                   |
| .-----------------------. |
| |                       | |
| '-----------------------' |
|---------------------------|
|   <  OK  >    <Cancel>    |
'---------------------------'
```

Come si vede dalla figura, viene richiesto l'inserimento del dominio NIS; questa informazione dipende da come è configurato il servente NIS a cui ci si vuole rivolgere e si può ottenere presso un elaboratore già configurato con il comando '**`nisdomainname`**', senza argomenti. Dopo l'inserimento e la conferma si passa all'indicazione dell'indirizzo IP del servente:

```
nlnx-domain OK
```

```
.--------NIS server--------.
| Please insert the NIS     |
| server IPv4 address:      |
| .-----------------------. |
| |                       | |
| '-----------------------' |
|---------------------------|
|  <  OK  >      <Cancel>   |
'---------------------------'
```

In generale è meglio inserire l'indirizzo anche se appartenente alla rete locale; si passa così all'inserimento dell'indirizzo del servente DNS e inizialmente viene proposto lo stesso usato per il NIS:

```
192.168.1.254 OK
```



```
.--------NFS server--------.
| Please insert the NFS     |
| server IPv4 address:      |
| .-----------------------. |
| |192.168.1.254          | |
| '-----------------------' |
|---------------------------|
|  <  OK  >      <Cancel>   |
'---------------------------'
```

```
192.168.1.254 OK
```

Se tutto funziona come si deve, viene innestata la directory '`/home/`' remota e viene avviato il servizio NIS per l'acquisizione delle utenze. A quel punto, le directory personali degli utenti comuni locali non sono più accessibili, perché nascoste sotto quanto acquisito dal servente NFS.

Volendo fare le stesse cose a mano, senza l'aiuto dello script, si potrebbe ottenere lo stesso risultato dell'esempio attraverso i passaggi seguenti:

1. si modifica o si crea il file '`/etc/defaultdomain`', inserendo una riga contenente esattamente il nome del dominio NIS;

   ```
   nlnx-domain
   ```

2. si modifica o si crea il file '`/etc/yp.conf`', inserendo una riga contenente la direttiva '**`ypserver`**', con l'indirizzo del servente NIS;

   ```
   ypserver 192.168.1.254
   ```

3. si innesta la directory '`/home/`' remota;

   ```
   # mount -t nfs 192.168.1.254:/home /home [Invio]
   ```

4. si riavvia il servizio NIS.

   ```
   # /etc/init.d/nis stop [Invio]

   # /etc/init.d/nis start [Invio]
   ```

Per approfondire l'argomento si possono consultare le sezioni 36.3 e 36.4.

Si osservi che l'utente '**`tizio`**' e gli altri utenti comuni standard, previsti per NLNX, sono associati a numeri UID e GID inferiori a 1000, ovvero al di sotto del livello iniziale previsto per le utenze comuni. Il NIS predisposto con NLNX prevede la condivisione delle utenze che abbiano numeri UID da 1000 in su, pertanto, anche volendo, le utenze standard di NLNX non sono condivisibili se non si cambiano i numeri UID e GID nei file '`/etc/passwd`' e '`/etc/group`'. Tuttavia, il fatto che l'edizione standard di NLNX contenga sempre queste utenze speciali, fa sì che possano condividere gli stessi dati se si attiva un servizio NFS.

Controllo dello spazio utilizzato

NLNX è organizzato per la gestione delle quote di utilizzo della memoria di massa per gli utenti (non vengono considerati i gruppi), ma per l'attivazione di questa occorre procedere inizialmente attraverso comandi manuali. Vengono riassunte brevemente le fasi da eseguire nei punti successivi, partendo dal presupposto che il kernel sia in grado di gestire tale funzionalità.

1. Per prima cosa va controllato il file '`/etc/fstab`', nel quale deve apparire l'opzione '**`usrquota`**' per le unità da tenere sotto controllo. Nel caso si debba sottoporre alla gestione delle quote la stessa unità che si trova innestata nella directory '`/`', occorre creare un punto di innesto alternativo, come già prevede normalmente NLNX:

   ```
   /dev/sda4  none           swap   sw                                0  0
   /dev/sda2  /              auto   defaults,usrquota,errors=remount-ro  0  1
   /dev/sda2  /RO-FS/RW-FS   auto   defaults,usrquota,errors=remount-ro  0  0
   proc       /proc          proc   defaults                          0  0
   none       /proc/bus/usb  usbfs  defaults                          0  0
   sys        /sys           sysfs  defaults                          0  0
   …
   ```

2. Se il file ‘`/etc/fstab`’ viene modificato per unità che risultano essere già innestate, occorre procedere a un reinnesto, in modo che l’opzione ‘**`usrquota`**’ venga acquisita correttamente. L’esempio segue quanto già visto nell’estratto di file ‘`/etc/fstab`’:

   # **`mount -o remount /`** [*Invio*]

   # **`mount -o remount /RO-FS/RW-FS`** [*Invio*]

3. Quando le unità dispongono correttamente dell’opzione ‘**`usrquota`**’, va fatta la prima scansione per la creazione dei file ‘`auser.quota`’. Se si tratta di un file system che attualmente risulta utilizzato in lettura e scrittura, può essere necessaria l’opzione ‘**`-m`**’:

   # **`quotacheck -u -v -c -m -a`** [*Invio*]

4. Una volta fatta la scansione, si attiva la gestione delle quote con lo script ‘`/etc/init.d/quota`’:

   # **`/etc/init.d/quota start`** [*Invio*]

   Naturalmente, all’arresto e al riavvio, questa operazione viene svolta automaticamente.

L’ultima fase consiste nell’attribuire effettivamente le quote agli utenti, per esempio tramite ‘**`setquota`**’, ma questa operazione va svolta effettivamente attraverso ‘**`nlnxrc quota set`**’:

`nlnxrc quota set` [*Invio*]

Quando si avvia il comando per la prima volta, viene creato il file ‘`/etc/nlnx/HOME_DISK_SPACE_ALLOWED`’, con il contenuto seguente, assegnando una disponibilità di spazio pari a 10000000 byte; contestualmente viene avviato un programma per la modifica dei file di testo, allo scopo di poter cambiare il contenuto di questo file:

```
#
# USER:MAX_BYTES
#
# the record without user name is the default:
# :MAX_BYTES
#
:10000000
```

Come si vede dal suggerimento nel commento iniziale, si può specificare il limite per ogni utente, con la forma:

utente **:** ***n_byte***

Per dare un limite a tutti gli utenti non dichiarati espressamente in questo file occorre la direttiva seguente:

: ***n_byte***

Quando si salva il file e si termina il funzionamento del programma di modifica dei file di testo, il controllo ritorna a ‘**`nlnxrc`**’ che chiede se si vuole procedere con l’attribuzione delle quote, secondo quanto definito nel file. Se la risposta è affermativa, le quote vengono assegnate o modificate.

Per la precisione, le quote vengono assegnate relativamente allo spazio utilizzato, senza considerare la quantità di file (numeri inode), fissando lo stesso valore per la quota e il limite massimo (*hard*). Tuttavia, il valore esatto è maggiore rispetto a quello richiesto effettivamente; per la precisione viene assegnato il 25 % in più.

Il limite massimo è pari al limite della quota, per evitare che gli utenti si trovino improvvisamente a perdere dei file quando scade il tempo di riserva (il tempo di grazia). In pratica, se il limite è stato superato, l’operazione di scrittura fallisce e l’utente ne è subito consapevole.

Va osservato anche che la richiesta di un valore di quota troppo elevato e non gestibile, si conclude nella non attribuzione di alcun limite nella quota. D’altra parte, assegnando il valore zero, si ottiene esattamente di non attribuire alcun limite di utilizzo della memoria di massa.

La configurazione delle quote attraverso ‘**`nlnxrc quota set`**’ è importante perché consente di attribuire la quota predefinita agli utenti che vengono aggiunti al sistema con il comando ‘**`nlnxrc user add`**’.

Controllo del numero di pagine stampabili e dell’origine delle stampe

Nella directory ‘`/etc/script/`’ sono presenti due script denominati ‘**`lpr`**’ e ‘**`lp`**’, il cui scopo è quello di eseguire un controllo preliminare su ciò che viene inviato alla stampa, per poi passare il compito ai programmi corrispondenti, che però si trovano nella directory ‘`/usr/bin/`’. La directory ‘`/etc/script/`’ si trova per prima nell’elenco dei percorsi di avvio (la variabile di ambiente ‘**`PATH`**’), così che gli script contenuti al suo interno vengono eseguiti prima di cercare programmi con lo stesso nome in altre collocazioni.

Gli script denominati ‘**`lpr`**’ e ‘**`lp`**’ controllano se è stato dichiarato qualcosa a proposito della volontà di limitare le stampe a un certo numero di pagine; se questo controllo non è stato richiesto, avviano i loro programmi omonimi; altrimenti, elaborano il file pervenuto per la stampa con lo script ‘**`print-filter`**’.

Lo script ‘**`print-filter`**’ che si trova sempre nella directory ‘`/etc/script/`’, cerca di riconoscere il file e se può, cerca di contarne le pagine. Se non può riconoscere il file, o comunque se non può elaborarlo, lo restituisce tale e quale (in tal caso il file in questione continua il suo percorso normale verso la stampa); se invece può gestirlo, lo rielabora in modo da contarne le pagine: se si determina che le pagine da stampare non superano il limite stabilito con la configurazione, il file viene inviato alla stampa; altrimenti viene annotato un messaggio di errore nel registro del sistema.

È evidente che questo meccanismo di controllo si può raggirare facilmente, utilizzando i programmi ‘**`lpr`**’ e ‘**`lp`**’ con il loro percorso: ‘`/usr/bin/lpr`’, ‘`/usr/bin/lp`’. Ma lo scopo di questo sistema è solo quello di evitare degli errori, ovvero l’invio di una stampa non desiderata con una quantità enorme di pagine.

La configurazione con la quale si fissa la quantità massima di pagine per stampa, spetta solo all’amministratore:

`nlnxrc print maxpages` [*Invio*]

```
.-------------Set max lpr/lp printable pages---------------.
| Please insert how many pages are allowed to be printed   |
| for any single print; if you enter zero or you leave     |
| blank, there is no limit:                                |
| .------------------------------------------------------. |
| |                                                      | |
| `------------------------------------------------------' |
|----------------------------------------------------------|
|               <  OK  >          <Cancel>                |
`----------------------------------------------------------'
```

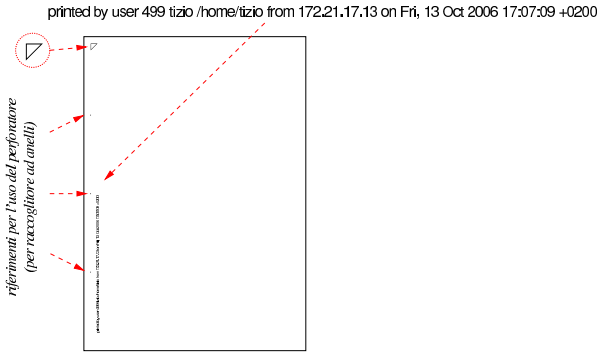
Come si vede dal suggerimento, lasciando il campo vuoto o inserendo esplicitamente il valore zero, si ottiene di annullare il controllo, in modo da non avere alcun limite di stampa. In questo caso vengono specificate 11 pagine:

`11` OK

```
.-----------------------------------------.
| It will be allowed to print 11 pages at |
| a time.                                 |
|                                         |
|-----------------------------------------|
|              <  OK  >                   |
`-----------------------------------------'
```

Se il controllo delle pagine da stampare viene eseguito, lo script ‘**`print-filter`**’ aggiunge una specie di «timbro», con le informazioni dell’origine della stampa. In condizioni normali, si tratta di

una scritta verticale, che appare vicino al bordo inferiore sinistro del foglio, come si vede nell'immagine successiva:



In particolare, se si tratta di file di testo, lo script '**`print-filter`**' li impagina in modo particolare, mettendo l'informazione sull'origine della stampa in alto. Eventualmente, si può intervenire nello script (collocato nella directory '`/etc/script/`') per cambiare l'impaginazione.

Utenze speciali per l'amministrazione

Per facilitare la gestione di un laboratorio, o di una rete locale che si affida a NLNX, è possibile creare delle utenze amministrative speciali, a cui viene associato lo script '`/etc/script/ADMIN`', attraverso la mediazione del programma '`/etc/script/admin`' che controlla alcune cose prima di acquisire i privilegi di un super-utente con UID pari a zero. La creazione di tali utenze è riservata all'utente '**`root`**', il quale procede con il comando '**`nlnxrc admin add`**', in modo analogo a quanto già descritto a proposito degli utenti comuni, con la differenza che il nome di tale amministratore deve essere più corto.

L'utente amministratore che si crea in questo modo, predispone un'utenza Unix con numero UID superiore a 60000, appartenente al gruppo '**`admin`**', pari al numero 60999. Ma oltre a questa, crea un'altra utenza Unix denominata '**`win.`*nome***', con UID pari a zero, senza una parola d'ordine valida, rendendola inservibile dal lato Unix, ma tale utenza viene invece aggiunta anche alla gestione di Samba, per consentire all'utente '**`win.`*nome***' di aggiungere un elaboratore con MS-Windows alla gestione delle utenze di Samba.

Supponendo di avere creato l'utenza amministrativa '**`rossi`**', l'accesso con questa al sistema offre un menù di funzioni:

```
login: rossi [Invio]

Password: digitazione_all'oscuro [Invio]
```

Il menù di funzioni cambia a seconda del contesto in cui viene avviato; per esempio, c'è differenza se si tratta di un elaboratore cliente o del servente che gestisce le utenze, inoltre c'è differenza se l'accesso proviene dalla console o da un collegamento remoto. Quello che segue è il menù che si ottiene da un collegamento remoto al servente che gestisce le utenze complessive della rete locale:

```
.-----------------------Admin menu-----------------------.
| Admin limited menu                                     |
| .----------------------------------------------------. |
| |   adduser         Add a new user                   | |
| |   passwd          Change a user's password         | |
| |   description     Change a user's description      | |
| |   user info       Show user info                   | |
| |   home info       Show home directory info         | |
| |   quota report    Show sorted user's quota         | |
| |   nis-make        Rebuild NIS database             | |
| |   add machine     Add a new MS-Windows computer    | |
| |   proxy access    HTTP proxy access permissions    | |
| |   admin passwd    Change your password             | |
| |   custom1         Custom script 1                  | |
| |   custom2         Custom script 2                  | |
| |   custom3         Custom script 3                  | |
| |   exit            Quit                             | |
| '----------------------------------------------------' |
|--------------------------------------------------------|
|               <  OK  >         <Cancel>               |
'--------------------------------------------------------'
```

Tabella u32.24. Descrizione di alcune delle funzioni disponibili.

Funzione	Descrizione
`adduser`	Consente di aggiungere un'utenza al sistema.
`passwd`	Consente di cambiare la parola d'ordine di un'utenza.
`description`	Consente di modificare la descrizione di un'utenza.
`user info` `home info`	Consente di scorrere l'elenco degli utenti, in ordine di nominativo oppure in ordine di directory personale, per ottenere poi alcune informazioni salienti sull'utenza relativa.
`quota report`	Se è attiva la gestione delle quote, mostra l'elenco delle quote degli utenti, ordinata in modo decrescente per quantità di spazio utilizzato.
`clock`	Consente di cambiare la data e l'ora.
`nis-make`	Ricostruisce la base di dati NIS a partire dalle informazioni aggiornate delle utenze.
`nis restart`	Riavvia i servizi NIS (servente o cliente, a seconda della configurazione).
`lpd restart`	Riavvia il servente di stampa.
`printer access`	Controlla l'accesso al servente di stampa.
`gpm restart`	Riavvia il demone per la gestione del mouse.
`udev restart`	Riavvia la gestione automatica dei file di dispositivo.
`print maxpages`	Limita le stampe a un numero massimo di pagine per volta.
`lprm`	Elimina la coda di stampa.
`proxy access`	Controlla l'accesso al proxy HTTP.
`add machine`	Aggiunge un'utenza speciale, corrispondente a un elaboratore MS-Windows, per consentirgli l'accesso con Samba.
`admin passwd`	Consente all'amministratore di cambiarsi la propria parola d'ordine.
`reboot`	Esegue il riavvio degli elaboratori (devono essere selezionati).
`shutdown`	Avvia lo spegnimento degli elaboratori (devono essere selezionati).
`custom`***n***	Avvia lo script '**`CUSTOM`*n***' che deve essere presente nella directory '`/etc/script/`'. Si tratta di script personalizzati.
`exit`	Esce dal menù.

Utenze condivise e configurazione automatica con NLNX

Condivisione delle utenze attraverso NIS, NFS e Samba 151
Utilizzo di SSHfs al posto di NFS 152
Gestione delle utenze 153
Samba 154
Le utenze di Samba 157
Servente DHCP 157

`smb.conf` 154

Quando si installa NLNX può essere conveniente predisporre un servizio di condivisione delle utenze e delle directory personali, assieme a un meccanismo di configurazione automatica, utile in particolare per gli elaboratori avviati con un DVD *live* o comunque con un file system in sola lettura.

Tabella u16.4. Script '**`nlnxrc`**': configurazione del servizio DHCP.

Comando	Descrizione
`nlnxrc dhcp-server config` `nlnxrc dhcp-server unconf`	Attiva o disattiva il funzionamento in qualità di servente DHCP.
`nlnxrc dhcp-server edit`	Modifica la configurazione del servente DHCP, intervenendo nel file di configurazione in modo libero.

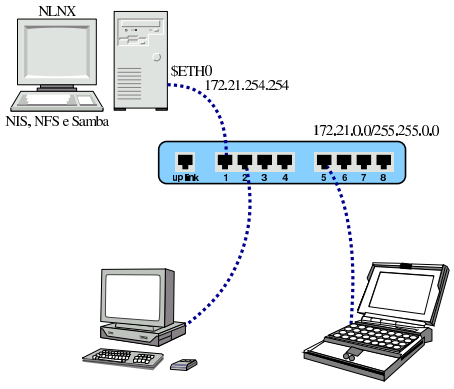
Condivisione delle utenze attraverso NIS, NFS e Samba

Attraverso i protocolli NIS e NFS, NLNX può consentire la condivisione delle utenze e delle directory personali in una rete locale; inoltre, attraverso Samba, le directory personali degli utenti sono accessibili anche attraverso sistemi MS-Windows.

Si osservi che la procedura prevista per NLNX richiede che sia il NIS, sia le directory personali da condividere con il protocollo NFS e attraverso Samba, si trovino nello stesso elaboratore; inoltre è indispensabile che le directory personali si articolino a partire da '`/home/`'.

La procedura predisposta da NLNX prevede la pubblicazione del contenuto dei file '`/etc/passwd`', '`/etc/shadow`' e '`/etc/group`', ignorando ogni altro file che il NIS potrebbe fornire.

Figura u33.1. NIS, NFS e Samba offerti da un elaboratore in cui è installato NLNX.



Per attivare il NIS in modo da offrire alla rete locale la condivisione dei file '`/etc/passwd`', '`/etc/shadow`' e '`/etc/group`', si procede come segue:

```
# nlnxrc nis-server config [Invio]
```

```
.--------NIS domain----------.
| Please insert the NIS      |
| domain:                    |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Il dominio NIS viene deciso liberamente, in questo caso scegliendo il nome '**nis.domain**':

```
nis.domain OK
```

```
.--------NIS server----------.
| Please insert the NIS      |
| server IPv4 address:       |
| .------------------------. |
| |172.21.254.254          | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Si presume che la rete interna sia già stata configurata, pertanto l'indirizzo viene proposto in modo automatico:

```
172.21.254.254 OK
```

Un risultato equivalente avrebbe potuto essere ottenuto attraverso le opzioni di avvio seguenti:

```
n_nis_domain=nis.domain n_nis_server=172.21.25
```

Naturalmente, la condivisione delle informazioni contenute nei file delle utenze non è sufficiente: occorre condividere anche le directory personali degli utenti. Se la configurazione di NLNX non è stata modificata, la directory '`/home/`' risulta accessibile a qualunque nodo di rete con indirizzi IPv4 privati.

Prima di questo capitolo è spiegato in che modo configurare un nodo cliente per servirsi di un servizio NIS e NFS. In quel caso, è possibile distinguere i due serventi, mentre se si offre il servizio con NLNX, la procedura richiede che entrambi i servizi risiedano assieme nello stesso elaboratore (presso lo stesso indirizzo IPv4).

Si osservi che per motivi pratici, lo script '`/etc/profile`' di NLNX, se riesce a determinare dalla configurazione che il proprio elaboratore dovrebbe ricoprire il ruolo di servente NIS, rifiuta l'accesso degli utenti comuni.

Il file di configurazione '`/etc/default/nis`' viene modificato automaticamente dallo script '**nlnxrc**'; tuttavia, se si vuole evitare che il servente NIS metta in funzione il demone '**ypbind**' (che procura una serie di inconvenienti), è bene aggiungere la riga seguente in quel file:

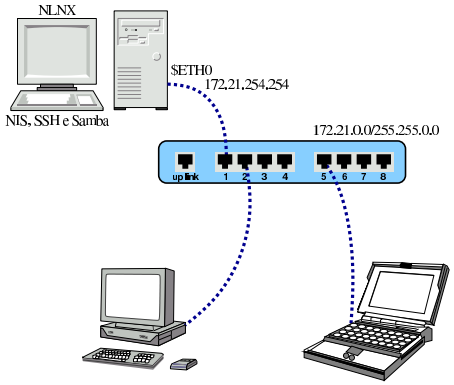
```
…
NISCLIENT=false
…
```

Se questa riga è presente, viene gestita correttamente da '**nlnxrc**', anche quando si configura il funzionamento come cliente NIS.

Utilizzo di SSHfs al posto di NFS

Quando un utente tenta di accedere attraverso il sistema grafico, se la sua directory personale non è presente e invece è stato configurato il riferimento a un servente NIS o NFS, viene tentato un collegamento a tale servente con l'ausilio di SSHfs. Pertanto, da una finestra di terminale viene chiesto ulteriormente di fornire la parola d'ordine per accedere, quindi viene concesso di accedere alla propria directory personale attraverso un tunnel SSH.

Figura u33.5. NIS, SSH e Samba offerti da un elaboratore in cui è installato NLNX.



Infatti, il servizio NFS per la condivisione delle directory personali degli utenti, gestito da NLNX, è quello della versione 3, non cifrato, il quale può andare bene in una piccola rete locale, in cui non ci sia il pericolo di comportamenti scorretti da parte degli utenti. Pertanto, quando non ci si può accontentare, presso il servente è meglio disabilitare la condivisione della directory '`/home/`' tramite il servizio NFS, lasciando così che le connessioni alle directory personali avvengano tramite SSHfs.

Si osservi, comunque, che l'accesso ai propri dati personali tramite SSHfs, non è equivalente a quello che si ha con un servizio NFS, perché si vedono esclusivamente i propri dati. Per esempio, se con NFS viene condivisa una directory (a partire da '`/home/`') con i permessi per tutti gli utenti, questa directory risulterebbe inaccessibile. A tale proposito, presso il servente è bene condividere la directory '`/data/`' (con NFS), per collocare lì i dati che devono essere disponibili a tutti e per i quali non ci sono problemi di segretezza.

Gestione delle utenze

Quando si attiva un servente NIS-NFS, è necessario gestire le utenze esclusivamente nell'elaboratore che offre questo servizio. In generale, una volta installato NLNX secondo la modalità normale, si potrebbero utilizzare gli strumenti consueti per tale gestione, ma è meglio avvalersi in ogni caso dello script '**nlnxrc**':

```
nlnxrc user add
```

```
nlnxrc user del [utente]
```

```
nlnxrc user passwd
```

```
nlnxrc user description
```

```
nlnxrc user info
```

La sintassi dovrebbe essere già comprensibile così: '**add**' aggiunge un utente; '**del**' lo elimina, assieme alla sua directory personale; '**passwd**' consente di cambiargli la parola d'ordine per accedere; '**description**' consente di cambiarne la descrizione; '**info**' consente di ottenere le informazioni sintetiche di un'utenza.

Per motivi pratici, la directory personale dell'utente che viene creato contiene nel percorso un'informazione aggiuntiva che, in caso non sia specificata e non siano presenti altri esempi del genere, è costituita dall'anno di creazione, per individuare in modo molto semplice le utenze più vecchie, senza bisogno di interrogare il file '`/etc/shadow`'. Per esempio, se nell'anno 2012 si crea l'utenza '**pippo**' e si segue ciò che viene proposto, si ottiene la directory personale '`/home/2012/pippo/`'.

L'utilizzo del comando '**`nlnxrc user add`**' ha anche il vantaggio di facilitare l'inserimento di più utenze, dal momento che alla fine di ogni inserimento ne viene proposto subito un altro (che comunque può essere annullato); inoltre, al termine degli inserimenti viene riallineato il NIS.

Samba

La configurazione di Samba non viene gestita tramite '**`nlnxrc`**', ma rimane da sistemare a mano. Il file predefinito ha l'aspetto seguente, commenti inclusi, e ha lo scopo di consentire l'accesso ai propri dati personali:

```
[global]
    server string = default NLNX configuration
    workgroup = NLNX
    netbios name = nlnx00
    local master = no
    domain master = no
#   local master = yes
#   domain master = yes
#   preferred master = yes
#   os level = 64
#   domain logons = yes
#   logon path = \\%L\profiles\%u
#   wins support = yes
#   time server = yes
    security = user
    hostname lookups = no
    hosts allow = 127.0.0.0/8 10.0.0.0/8 172.16.0.0/12 ↩
↪192.168.0.0/16
    dns proxy = no
    log file = /var/log/samba/log.%m
    max log size = 1000
    log level = 3
    encrypt passwords = yes
    smb passwd file = /etc/samba/smbpasswd
    passdb backend = smbpasswd:/etc/samba/smbpasswd
    socket options = TCP_NODELAY
    hide files = /Desktop/Mail/mail/dosemu/
#   load printers = yes
#   printing = bsd
#   printcap name = /etc/printcap
    unix password sync = yes
    passwd program = /usr/bin/passwd %u
    pam password change = yes
[homes]
    comment = home directories
    browseable = no
    writable = yes
    create mask = 0755
    directory mask = 0755

#[netlogon]
#   path = /home/.samba/netlogon
#   writeable = no
#   browseable = no
#   guest ok = yes

#[profiles]
#   path = /home/.samba/profiles
#   browseable = no
#   writeable = yes
#   create mask = 0600
#   directory mask = 0700

#[examples]
#    comment = examples
#    browseable = yes
#    path = /home/examples
#    writable = no
#    public = yes

#[printers]
#   comment = All Printers
#   browseable = no
#   path = /tmp
#   printable = yes
#   public = no
#   writable = no
#   create mode = 0700
```



```
#[print$]
#   comment = printer drivers
#   path = /var/lib/samba/printers
#   browseable = yes
#   read only = yes
#   guest ok = no
```

Nel file appaiono delle direttive commentate, allo scopo di avere un'idea delle voci che si potrebbero aggiungere. Le direttive evidenziate sono quelle che vanno modificate quasi sicuramente. In ogni caso, se si installa NLNX su più di un elaboratore è bene che le prime direttive siano differenti; inoltre, se l'installazione prevede la concentrazione delle utenze e delle directory personali, è bene che Samba sia in funzione solo in quello che gestisce effettivamente le utenze.

Per accedere alle directory personali, attraverso un sistema MS-Windows, gli utenti devono seguire una procedura che varia in funzione della versione di tale sistema operativo. Quello che si vede negli schemi successivi è una semplificazione che dovrebbe consentire di comprendere il procedimento, adattandolo poi alla realtà del proprio sistema.

Figura u33.7. Aggiunta di una risorsa di rete.

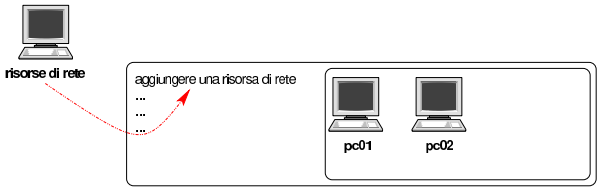


Figura u33.8. Indicazione del percorso della risorsa. Si suppone che l'elaboratore in cui è in funzione NLNX con il servizio Samba per la condivisione delle directory personali sia raggiungibile all'indirizzo IPv4 192.168.1.253. Inoltre, l'utente ipotetico è denominato «rossimario».

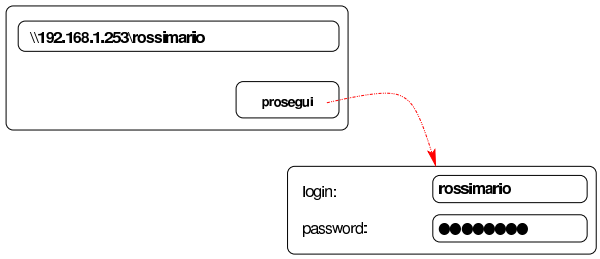
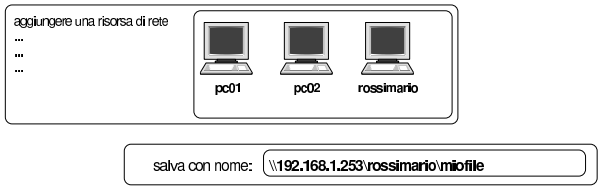


Figura u33.9. La risorsa risulta disponibile ed è possibile accedervi come se fosse un'unità a disco. Una volta collegata la risorsa, si suppone di voler salvare al suo interno un file con il nome '`miofile`'.



È molto probabile che il sistema MS-Windows chieda di memorizzare la parola d'ordine inserita: è evidente che ciò non va fatto, altrimenti un estraneo potrebbe accedere conoscendo semplicemente il nominativo-utente. Inoltre, al termine dell'utilizzo della risorsa, è necessario procedere al suo distacco, come si farebbe con un'unità rimovibile, altrimenti i dati rimarrebbero accessibili.

Le figure successive mostrano il procedimento in un sistema MS-Windows 7.

Figura u33.10. Per connettersi a una risorsa di rete è necessario selezionare la voce *Connetti unità di rete* contenuta nel menù che si ottiene premendo il tasto destro sulla voce *Computer*.

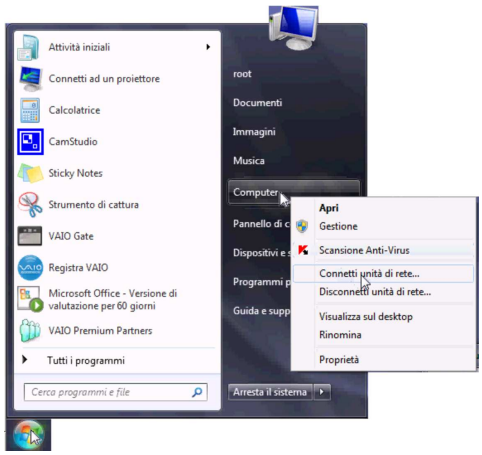


Figura u33.11. Si deve quindi specificare il percorso per raggiungere la propria cartella; in questo caso si tratta dell'elaboratore 172.17.1.254 e la cartella ha il nome '`giacominidaniele`'.

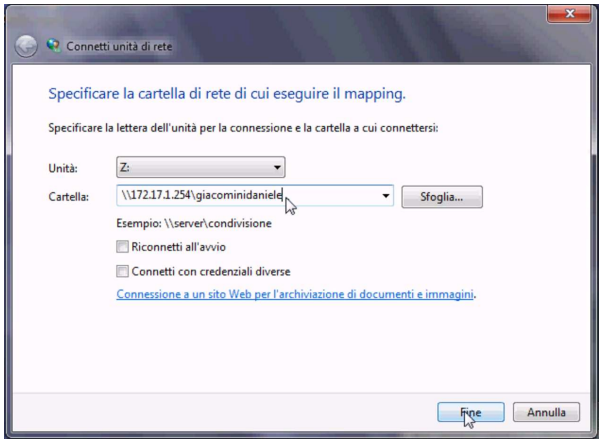


Figura u33.12. Se il percorso specificato esiste, viene richiesto di identificarsi.

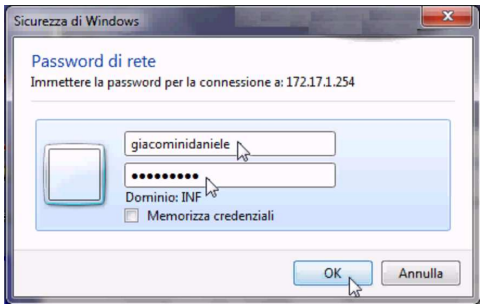
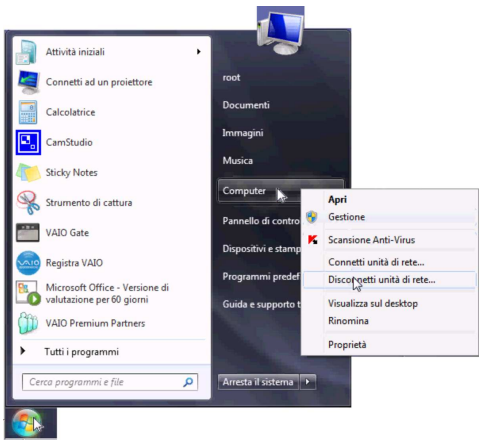


Figura u33.13. Successivamente, per chiudere la connessione con un'unità di rete occorre utilizzare nuovamente il menù che si ottiene facendo un clic con il tasto destro del mouse sopra la voce *Computer*.



Le utenze di Samba

Samba utilizza il file '`/etc/samba/smbpasswd`' per tenere traccia degli utenti. Questo file è incompatibile con '`/etc/passwd`' e per mantenere le utenze Unix allineate con quelle del protocollo SMB è necessario utilizzare i comandi di '**`nlnxrc`**':

```
nlnxrc user add
```

```
nlnxrc user del [utente]
```

```
nlnxrc user passwd
```

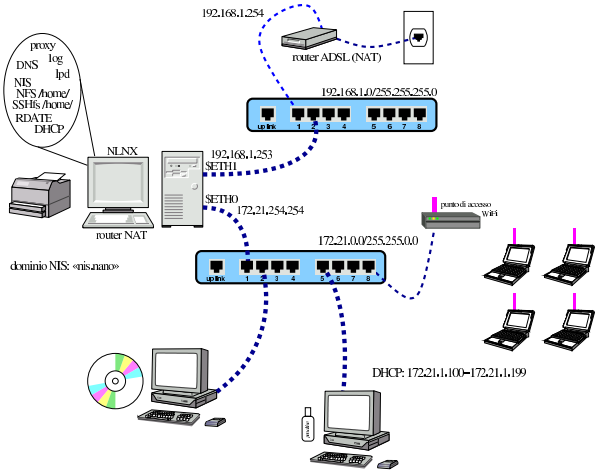
Servente DHCP

In condizioni normali, prima che sia stata inserita una propria configurazione per l'utilizzo della rete, NLNX utilizza il protocollo DHCP per tentare di configurarsi in modo automatico. Tale configurazione automatica, se le informazioni sono disponibili, si spinge anche all'uso del NIS, della condivisione delle directory personali, della stampa condivisa in rete, della gestione di un registro complessivo.

In generale, l'uso predefinito del protocollo DHCP serve soprattutto per facilitare il funzionamento di NLNX da unità in sola lettura, quando è già disponibile tale servizio e non si deve fare un lavoro specifico. Quando invece si vuole usare il DHCP anche per la condivisione delle utenze e gli altri servizi, diventa indispensabile attivare il proprio servente DHCP.

NLNX è pensato in modo particolare per il contesto di una rete usata a scuola, dove è molto facile entrare in conflitto con altri servizi DHCP, probabilmente di altri sistemi operativi. Il servente DHCP di NLNX può essere attivato in un elaboratore che funge da router, ma se si vuole isolare il servizio DHCP rispetto alle altre reti, è indispensabile che l'elaboratore che svolge il lavoro sia provvisto di **due interfacce di rete**, una rivolta alla rete locale da servire e una rivolta all'esterno, o al resto della rete del complesso, per evitare queste interferenze. Pertanto, se non è possibile isolare la propria rete attraverso un router composto da due interfacce ed esistono altri servizi DHCP accessibili, è meglio evitare l'uso del DHCP con NLNX.

Figura u33.14. La situazione prevista, con esempi di indirizzi, per l'utilizzo di un servente DHCP con NLNX.



In base all'esempio mostrato nella figura, si può procedere alla configurazione del servente DHCP nel modo seguente:

```
# nlnxrc dhcp-server config [Invio]
```

```
.-----------------------DHCP range-------------------------.
| Current "/etc/dhcp3/dhcpd.conf" file might be set as it  |
| follows:                                                 |
|                                                          |
| ddns-update-style none;                                  |
| subnet 172.21.0.0 netmask 255.255.0.0 {                  |
| #  range 172.21.254.100 172.21.254.199;                  |
|   option broadcast-address 172.21.255.255;               |
|   option routers 172.21.254.254;                         |
|   option domain-name-servers 172.21.254.254;             |
|   option time-servers 172.21.254.254;                    |
|   option ntp-servers 172.21.254.254;                     |
|   option root-path "172.21.254.254:/";                   |
|   option nis-domain "nis.nano";                          |
|   option nis-servers 172.21.254.254;                     |
|   option lpr-servers 172.21.254.254;                     |
|   option log-servers 172.21.254.254;                     |
| }                                                        |
|                                                          |
| Please insert or confirm the DHCP address range:         |
| .------------------------------------------------------. |
| |172.21.254.100 172.21.254.199                         | |
|-'------------------------------------------------------'-|
|                 <  OK  >          <Cancel>               |
'----------------------------------------------------------'
```

Dal momento che è stato stabilito di usare un intervallo di indirizzi differente per il DHCP, il valore viene cambiato:

[*Canc*][*Canc*]…

172.21.1.100 172.21.1.199 OK

Successivamente viene chiesto se si vuole definire qualche abbinamento fisso tra indirizzi Ethernet e indirizzi IPv4. Come si intuisce dall'esempio che viene mostrato, gli indirizzi IPv4 che si associano sono diversi dal pacchetto attribuito in modo dinamico:

```
172.21.1.1  00:0B:6A:64:8C:F7
172.21.1.2  00:0B:6A:64:8C:F8
172.21.1.3  00:0B:6A:64:9C:A3
```

Per l'inserimento di questi indirizzi si usa un programma di modifica di file di testo e le variazioni vanno salvate lasciando il nome predefinito del file temporaneo utilizzato. Al termine viene mostrato un sunto della configurazione attuale, da confermare:


```
.--------------DHCP server configuration----------------.
| Is the following configuration correct?                |
|                                                        |
| ddns-update-style none;                                |
| option option-128 code 128 = string;                   |
| option option-129 code 129 = text;                     |
| next-server 172.21.254.254;                            |
| subnet 172.21.0.0 netmask 255.255.0.0 {                |
|   range 172.21.1.100 172.21.1.199;                     |
|   option broadcast-address 172.21.255.255;             |
|   option routers 172.21.254.254;                       |
|   option domain-name-servers 172.21.254.254;           |
|   option time-servers pool.ntp.org;                    |
|   option ntp-servers pool.ntp.org;                     |
|   option root-path "172.21.254.254:/opt/NLNX";         |
|   option nis-domain "nis.domain";                      |
|   option nis-servers 172.21.254.254;                   |
|   option lpr-servers 172.21.254.254;                   |
|   option log-servers 172.21.254.254;                   |
|   use-host-decl-names      on;                         |
|   filename "/pxelinux/pxelinux.0";                     |
| }                                                      |
|--------------------------------------------------------|
|              <  Yes  >        <  No  >                |
'--------------------------------------------------------'
```

Se la configurazione proposta è quella che si desidera, si può confermare:

YES

Altrimenti si annulla, salvando ugualmente al configurazione, ma senza attivare immediatamente il servizio:

NO

Se la configurazione ottenuta non è quella desiderata (per esempio il dominio NIS non è quello voluto oppure alcuni servizi sono riferiti a indirizzi errati), conviene modificarla attraverso il comando seguente:

```
# nlnxrc dhcp-server edit [Invio]
```

Con questo si passa alla modifica diretta del file '`/etc/dhcp3/dhcpd.conf`'. Quando si salva, si riavvia il servizio DHCP. In base all'esempio, la configurazione ottenuta sarebbe quella seguente:

```
ddns-update-style none;
option option-128 code 128 = string;
option option-129 code 129 = text;
next-server 172.21.254.254;

subnet 172.21.0.0 netmask 255.255.0.0 {
  range 172.21.1.100 172.21.1.199;
  option broadcast-address 172.21.255.255;
  option routers 172.21.254.254;
  option domain-name-servers 172.21.254.254;
  option time-servers pool.ntp.org;
  option ntp-servers pool.ntp.org;
  option root-path "172.21.254.254:/opt/NLNX";
  option nis-domain "nis.domain";
  option nis-servers 172.21.254.254;
  option lpr-servers 172.21.254.254;
  option log-servers 172.21.254.254;
  use-host-decl-names      on;
  filename "/pxelinux/pxelinux.0";
  host 172.21.1.1 { hardware ethernet 00:0B:6A:64:8C:F7; fixed-address 172.21.1.1; }
  host 172.21.1.2 { hardware ethernet 00:0B:6A:64:8C:F8; fixed-address 172.21.1.2; }
  host 172.21.1.3 { hardware ethernet 00:0B:6A:64:9C:A3; fixed-address 172.21.1.3; }
}
```

Controllo dell'accesso a servizi HTTP esterni

Situazione tipica di utilizzo 161
Controllo a gruppi 161

Il servizio proxy HTTP di NLNX è costituito da OOPS.

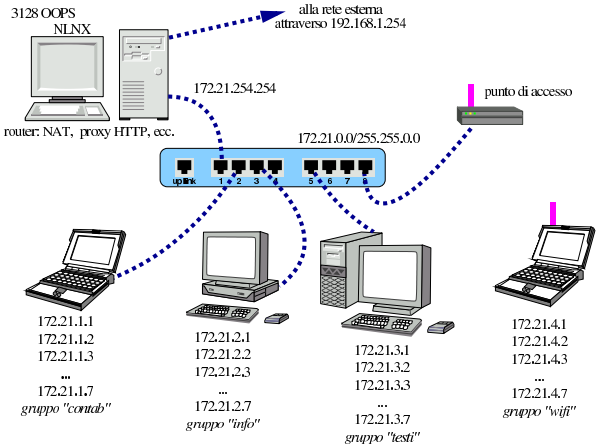
Tabella u16.5. Script '**`nlnxrc`**': configurazione del servizio proxy HTTP.

Comando	Descrizione
`nlnxrc proxy clients` [*gruppo*]	Definisce l'elenco dei nodi di rete previsti per l'accesso al proprio servizio proxy HTTP.
`nlnxrc proxy access` [*gruppo*]	Seleziona i nodi di rete, tra quelli previsti, che possono accedere effettivamente al servizio.

Situazione tipica di utilizzo

Nella figura successiva appare la schematizzazione di una rete locale composta da diversi gruppi di elaboratori, organizzati opportunamente in base agli indirizzi (172.21.1.∗, 172.21.2.∗, 172.21.3.∗, 172.21.4.∗), che utilizzano tutti il router 172.21.254.254, il quale ha anche la funzione di proxy HTTP.

Figura u34.1. Situazione tipica di utilizzo del servizio proxy HTTP di NLNX



Per prima cosa occorre distinguere se il servizio proxy HTTP è «normale» o trasparente, come viene configurato nel servente attraverso il comando '**`nlnxrc network config`**'. Logicamente, se si tratta di un servizio trasparente, tutti i nodi periferici che si devono avvalere del router, sono sottoposti al controllo del proxy HTTP, diversamente vanno configurati espressamente i programmi usati come navigatori.

Se il servizio è normale, nel senso che i programmi usati come navigatori devono essere configurati espressamente, se si configura la richiesta del servizio verso la porta 3128 si è sottoposti al controllo di OOPS. Quando invece il servizio proxy HTTP è trasparente, tutto il traffico che va verso l'esterno, alla porta 80, viene ridiretto automaticamente alla porta 3128, ovvero a OOPS. Tuttavia, se il servizio è trasparente, il traffico diretto volontariamente alla porta 3128, viene invece dirottato al servente HTTP locale.

Controllo a gruppi

In condizioni normali, il proxy HTTP di NLNX va configurato in modo trasparente, nel nodo che svolge il ruolo di router, per poter poi, attraverso il proxy, controllare l'accesso ai siti.

È possibile limitare l'accesso al proxy HTTP intervenendo su insiemi separati di nodi, in modo da poter controllare chi, nella propria rete locale, può accedere ai servizi HTTP esterni. Tali insiemi di nodi

vengono nominati arbitrariamente; per esempio il gruppo «contab» si crea e si aggiorna così:

```
# nlnxrc proxy clients contab [Invio]
```

Si inizia stabilendo l'elenco di nodi che si possono avvalere potenzialmente del proxy HTTP. Si devono indicare solo nodi singoli:

```
172.21.1.1
172.21.1.2
172.21.1.3
172.21.1.4
172.21.1.5
172.21.1.6
172.21.1.7
```

Gli altri gruppi si definiscono nello stesso modo:

```
# nlnxrc proxy clients info [Invio]

# nlnxrc proxy clients testi [Invio]

# nlnxrc proxy clients wifi [Invio]
```

È importante evitare di creare delle sovrapposizione tra i gruppi di accesso. Dopo la predisposizione degli elenchi, si può passare al controllo effettivo dell'accesso:

```
# nlnxrc proxy access contab [Invio]
```

```
.----------HTTP proxy access permissions-------------.
| Please, select or deselect who can access to the   |
| HTTP proxy:                                        |
| .------------------------------------------------. |
| | [ ] DENY_ALL       reset to no access allowed  | |
| | [ ] ALLOW_ALL      reset to all access allowed | |
| | [ ] 172.21.1.1     allow_172.21.1.1            | |
| | [ ] 172.21.1.2     allow_172.21.1.2            | |
| | [ ] 172.21.1.3     allow_172.21.1.3            | |
| | [ ] 172.21.1.4     allow_172.21.1.4            | |
| | [ ] 172.21.1.5     allow_172.21.1.5            | |
| | [ ] 172.21.1.6     allow_172.21.1.6            | |
| | [ ] 172.21.1.7     allow_172.21.1.7            | |
| '-----v(+)-----------------------------------------' |
|------------------------------------------------------|
|            <  OK  >         <Cancel>                |
'------------------------------------------------------'
```

Eventualmente, tra i nodi già definiti tramite il comando '**nlnxrc proxy access**' è possibile dichiarare quali escludere dal filtro svolto da DansGuardian:

```
# nlnxrc proxy filter contab [Invio]
```

```
.----------HTTP proxy content filter selection-------------.
| Please, select or deselect who can access to the HTTP    |
| proxy without content filtering:                         |
| .------------------------------------------------------. |
| | [ ] FILTER_ALL     reset to all filtered             | |
| | [ ] FREE_ALL       reset to all free to access any si| |
| | [*] 172.21.1.1     free_172.21.1.1                   | |
| | [*] 172.21.1.2     free_172.21.1.2                   | |
| | [*] 172.21.1.3     free_172.21.1.3                   | |
| | [ ] 172.21.1.4     free_172.21.1.4                   | |
| | [ ] 172.21.1.5     free_172.21.1.5                   | |
| | [ ] 172.21.1.6     free_172.21.1.6                   | |
| | [ ] 172.21.1.7     free_172.21.1.7                   | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|                <  OK  >         <Cancel>                 |
'----------------------------------------------------------'
```

In questo caso, con le selezioni che si vedono, si vuole fare in modo che i nodi con gli indirizzi 172.21.1.1, 172.21.1.2 e 172.21.1.3, siano esonerati dal filtro dei contenuti.

Quando si usano i comandi '**nlnxrc proxy** …' omettendo il nome del raggruppamento a cui si è interessati, se ne esiste già un solo non viene fatta alcuna richiesta e si passa a svolgere l'attività richiesta; se invece ne è disponibile più di uno, appare un menù da cui poter scegliere.

L'accessibilità di una stampante va limitato. NLNX consente di definire l'insieme degli elaboratori clienti che possono servirsi della stampante collegata al proprio.

Tabella u16.6. Script '**nlnxrc**': configurazione della stampa.

Comando	Descrizione
`nlnxrc printer config`	Consente di ricreare il file '`/etc/printcap`' impostando la coda di stampa predefinita per la stampante che si usa in un certo momento.
`nlnxrc print maxpages`	Consente di definire una quantità massima di pagine che possono essere stampate simultaneamente.
`nlnxrc printer clients` `nlnxrc printer access`	Nel primo caso consente di definire l'elenco dei nodi di rete previsti per l'accesso al proprio servizio di stampa; nel secondo consente di scegliere quali nodi, tra quelli previsti, possono accedere effettivamente al servizio di stampa.

La configurazione di una stampante locale comporta inizialmente che questa venga resa accessibile a chiunque, senza limitazioni, mentre la configurazione di una stampante remota coincide con la chiusura dell'accesso a chiunque, salvo ai programmi locali.[1] Per modificare questo sistema di massima, occorre procedere con due comandi di '**nlnxrc**':

```
# nlnxrc printer clients [Invio]
```

Per prima cosa occorre dichiarare quali sono i nodi da cui è prevista la possibilità di accedere al proprio servizio di stampa (locale o remoto che sia). In pratica, in questo elenco vanno inseriti gli indirizzi IPv4 di chi, per qualche ragione, deve avere la possibilità di stampare. Un elenco del genere potrebbe avere significato:

```
172.21.1.1
172.21.1.2
172.21.1.3
172.21.2.0/255.255.255.0
172.21.3.0/255.255.255.0
```

Come si vede, si possono indicare anche gruppi di nodi, specificando una maschera di rete; tuttavia, è bene che non si creino sovrapposizioni, altrimenti diventa poi difficile gestire il controllo dei permessi di accesso.

Supponendo di avere inserito esattamente l'elenco che si vede nell'esempio, dopo, con un altro comando di '**nlnxrc**', è possibili stabilire chi può accedere tra questi:

```
# nlnxrc printer access [Invio]
```

```
.--------------------Printer access permissions----------------------.
| Please, select or deselect allowed access to printer:               |
| .-------------------------------------------------------------. |
| | [ ] DENY_ALL                 reset to no remote access allowed | |
| | [ ] ALLOW_ALL                reset to all access allowed       | |
| | [ ] 172.21.1.1               allow_172.21.1.1                  | |
| | [ ] 172.21.1.2               allow_172.21.1.2                  | |
| | [ ] 172.21.1.3               allow_172.21.1.3                  | |
| | [ ] 172.21.2.0/255.255.255.0 allow_172.21.2.0/255.255.255.0    | |
| | [ ] 172.21.3.0/255.255.255.0 allow_172.21.3.0/255.255.255.0    | |
| '-----v(+)----------------------------------------------------' |
|---------------------------------------------------------------------|
|                  <  OK  >            <Cancel>                       |
'---------------------------------------------------------------------'
```

Le prime due voci sono costanti, le altre dipendono dall'elenco inserito in precedenza. Selezionando la voce '**ALLOW_ALL**' si ottiene di attivare tutte le voci previste, mentre '**DENY_ALL**' le disattiva tutte. Queste due voci iniziali servono solo per azzerare velocemente l'elenco e la loro selezione, fa sì che confermando la richiesta si ripre-

senti l'elenco, senza eseguire subito l'azione richiesta; pertanto, solo quando le prime due voci dell'elenco sono deselezionate si prende in considerazione la scelta di quelle sottostanti e viene aggiornata la configurazione.

Supponendo di abilitare l'accesso al gruppo costituito dagli indirizzi 172.21.2.∗, si può osservare cosa succede se si riavvia il comando la volta successiva:

```
.--------------------Printer access permissions-----------------------.
| Please, select or deselect allowed access to printer:               |
| .-------------------------------------------------------------------. |
| | [ ] DENY_ALL                 reset to no remote access allowed | | |
| | [ ] ALLOW_ALL                reset to all access allowed       | | |
| | [X] 172.21.2.0/255.255.255.0 allow_172.21.2.0/255.255.255.0    | | |
| | [ ] 172.21.1.1               allow_172.21.1.1                  | | |
| | [ ] 172.21.1.2               allow_172.21.1.2                  | | |
| | [ ] 172.21.1.3               allow_172.21.1.3                  | | |
| | [ ] 172.21.3.0/255.255.255.0 allow_172.21.3.0/255.255.255.0    | | |
| `-----v(+)----------------------------------------------------------' |
|---------------------------------------------------------------------|
|                 < OK >            <Cancel>                          |
`---------------------------------------------------------------------'
```

Al riavvio del comando, le voci che in precedenza erano state selezionate, appaiono all'inizio dell'elenco, così da non doverle cercare.

Dal punto di vista del risultato, quello che conta è l'elenco dei punti a cui è concesso accedere. Se nell'elenco sono state fatte delle duplicazioni, per esempio se appare il nodo 172.21.2.33 e anche 172.21.2.0/255.255.255.0, bloccando l'accesso al gruppo 172.21.2.∗, attraverso la voce 172.21.2.0/255.255.255.0, il nodo 172.21.2.33 rimane ammesso a inviare delle stampe. Naturalmente può darsi che questo sia ciò che si vuole; quel che conta è capirne la logica.

[1] In pratica, se nel proprio elaboratore si configura una coda di stampa diretta a un elaboratore remoto, è necessario che tale coda sia accessibile solo ai processi elaborativi locali, altrimenti, pur non disponendo di una stampante locale, altri elaboratori potrebbero contattare la propria coda e inviare, in pratica, stampe a quell'elaboratore remoto.

Mettere in comunicazione insegnanti e studenti

Organizzazione delle utenze 165
Funzionamento 165

NLNX prevede che si possano predisporre degli script personalizzati, denominati '**CUSTOM*n***' e collocati nella directory '`/etc/script/`'. Vengono però forniti degli esempi, in particolare '**CUSTOM1**', con il quale si copiano dei file tra insegnanti e studenti, per facilitarne il lavoro in un laboratorio didattico. Tuttavia, il funzionamento di tale script richiede che siano rispettate alcune convenzioni nella definizione delle utenze.

Le sezioni successive descrivono il contesto a cui è destinato lo script '**CUSTOM1**' fornito con NLNX e lo scopo che con questo si intende raggiungere.

Organizzazione delle utenze

Lo script '**CUSTOM1**' fornito con NLNX richiede che le directory personali destinate agli insegnanti siano collocate, a scelta, a partire da '`/home/DOCENTI∗/`' o '`/home/ITP∗/`'. Per esempio, il professor Sempronio Dicembrino potrebbe avere l'utenza '**dicembrinosempr**' e, di conseguenza, la directory personale '`/home/DOCENTI∗/dicembrinosempr/`' (oppure '`/home/ITP∗/dicembrinosempr/`' se si tratta di un insegnante tecnico-pratico).

Per gli studenti, invece, è necessario che la classificazione sia composta da: numero della classe, lettera maiuscola della sezione, lettera maiuscola (opzionale) per il corso, quattro cifre per l'anno scolastico. Per esempio, lo studente Mario Rossi della classe 5A, corso «I», nell'anno scolastico 2012/2013, avendo l'utenza '**rossimario**', deve avere la directory personale '`/home/5AI1213/rossimario/`'.

È evidente che tale organizzazione richieda uno spostamento manuale delle utenze degli studenti alla fine dell'anno scolastico, per farli passare alla classificazione dell'anno scolastico successivo (se si vuole procedere in questo modo, vanno spostate le directory personali e va modificato a mano il file '`/etc/passwd`').

Funzionamento

Quando si avvia lo script '**CUSTOM1**' la prima volta, questo predispone delle strutture di directory nelle cartelle personali di studenti e di docenti. In entrambi i casi, gli utenti trovano le directory '`verifiche/`' e '`strumenti/`'.

Gli studenti vedono nella directory '`verifiche/`' altre sottodirectory con i nomi corrispondenti a quelli degli insegnanti; mentre gli insegnanti, all'interno di '`verifiche/`' vedono delle directory con le suddivisioni in classi degli studenti (per esempio '`5AI1213/`'), e all'interno di quelle trovano delle directory con i nomi degli studenti rispettivi. In tal modo, lo studente che deve o vuole conferire a un insegnante dei file, li mette nella directory '`verifiche/`***docente***`/`'; così facendo, quell'insegnante può trovare i file nella directory '`verifiche/`***classe***`/`***studente***`/`'.

Analogamente, gli insegnanti vedono una struttura articolata in classi, a partire dalla directory '`strumenti/`'. Quando un insegnante mette dei file nella directory '`strumenti/`***classe***`/`', tutti gli studenti di quella classe trovano tali file nella loro directory '`strumenti/`***docente***`/`'.

Lo script '**CUSTOM1**' va avviato periodicamente, per garantire che i file vengano messi correttamente a disposizione di insegnanti e studenti, dato che questi vengono copiati tra le directory.

Per comprendere meglio il meccanismo, si prenda lo studente Mario Rossi, della classe 5Ci, la cui directory persona-

le corrisponde a ‘/home/5CI1213/rossimario/’ e l’insegnante Sempronio Dicembrino, la cui directory personale corrisponde a ‘/home/DOCENTI*/dicembrinosempr/’. Lo studente Rossi prepara una verifiche per l’insegnante Dicembrino, costituita dal file ‘esercizio’ che colloca nella propria directory ‘~/verifiche/dicembrinosempr/’; per converso, l’insegnante Dicembrino trova nella propria directory ‘~/verifiche/home/5CI1213/rossimario/’ il file ‘esercizio’.

Nello stesso modo, l’insegnante Dicembrino predispone un file, denominato ‘modello’ e lo mette nella propria directory ‘~/strumenti/5CI1213/’, perché gli studenti della classe 5Ci (dell’anno scolastico 2012/2013) possano utilizzarlo. Per converso, lo studente Rossi trova nella propria directory ‘~/strumenti/dicembrinosempr/’ il file ‘modello’ predisposto dall’insegnante.

Estendere il controllo delle utenze alla rete MS-Windows

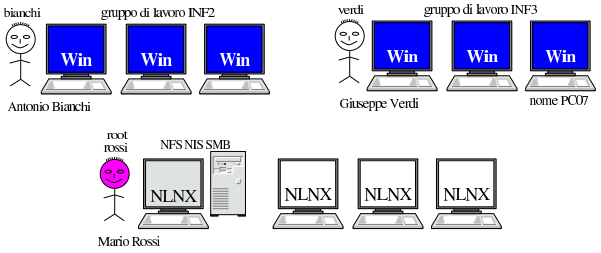
Situazione di esempio 167
Profili personali 168
Script di avvio 169
Utenze per gli elaboratori MS-Windows 169
Configurazione di MS-Windows XP Professional: disabilitazione della connessione cifrata 169
Configurazione di MS-Windows XP Professional: associazione al dominio 171
Configurazione di MS-Windows 7: definizione di due voci nel registro di sistema 172
Configurazione di MS-Windows 7: associazione al dominio 174
Spegnimento del servente NLNX 176
Servente WINS 176
Riferimenti 177

Attraverso Samba, NLNX può offrire, da elaboratori con sistema MS-Windows, la gestione delle cartelle personali. Oltre a questo, è possibile attivare la gestione delle utenze, in modo tale che anche dagli elaboratori MS-Windows sia richiesto di accedere specificando il nominativo utente e la parola d’ordine, come per gli altri elaboratori con NLNX. Per coordinare questa funzione occorre modificare manualmente la configurazione di Samba e provvedere alla creazione di alcuni utenti speciali.

Situazione di esempio

Per comprendere il meccanismo è necessario partire da un esempio, nel quale si ipotizza di disporre di una rete locale, unica sul piano fisico e anche sul piano logico, nel senso che gli indirizzi IPv4 devono essere tali da non richiedere il passaggio attraverso dei router.

Figura u37.1. Situazione di esempio, in cui si evidenziano tre persone con ruoli di amministrazione.



Nella figura si vede Mario Rossi che è il responsabile e l’amministratore del servente NLNX, oltre che di altri elaboratori clienti (sempre NLNX). Mario Rossi ha l’utenza amministrativa ‘**root**’, oltre a una seconda utenza amministrativa, denominata ‘**rossi**’, meno importante. Antonio Bianchi amministra il gruppo di lavoro ‘**INF2**’ e ha un’utenza amministrativa, presso il servente NLNX, denominata ‘**bianchi**’; Giuseppe Verdi amministra il gruppo di lavoro ‘**INF3**’ e ha un’utenza amministrativa, presso il servente NLNX, denominata ‘**verdi**’ (queste utenze amministrative sono state create con il comando ‘**nlnxrc admin add**’). Questi tre utenti, presso il servente NLNX, hanno anche delle utenze «normali», al pari di tutti gli altri utenti della rete.

La configurazione di Samba che si propone in questo contesto, contenuta nel file ‘/etc/smb.conf’ presso il servente NLNX, è quella nel listato successivo, dove sono evidenziate le voci salienti:

```
[global]
    server string = NLNX server
```

```
    workgroup = NLNX
    netbios name = nlnx00
    local master = yes
    domain master = yes
    preferred master = yes
    os level = 64
    domain logons = yes
    logon path = \\%L\profiles\%u
    wins support = yes
    time server = yes
    security = user
    hostname lookups = no
    hosts allow = 127.0.0.0/8 10.0.0.0/8 172.16.0.0/12
↪192.168.0.0/16
    dns proxy = no
    log file = /var/log/samba/log.%m
    max log size = 1000
    log level = 3
    encrypt passwords = yes
    smb passwd file = /etc/samba/smbpasswd
    passdb backend = smbpasswd:/etc/samba/smbpasswd
    socket options = TCP_NODELAY
    hide files = /Desktop/Mail/mail/dosemu/
    unix password sync = yes
    passwd program = /usr/bin/passwd %u
    pam password change = yes

[homes]
    comment = home directories
    browseable = no
    writable = yes
    create mask = 0755
    directory mask = 0755

[netlogon]
   path = /home/.samba/netlogon
   writeable = no
   browseable = no
   guest ok = yes

[profiles]
   path = /home/.samba/profiles
   browseable = no
   writeable = yes
   create mask = 0600
   directory mask = 0700
```

È necessario fare attenzione ai permessi della directory '`/home/.samba/profiles/`', con MS-Windows 7, può essere necessario dare i permessi di scrittura anche agli utenti diversi dal proprietario e dal gruppo:

```
# chmod o+rw /home/.samba/profiles [Invio]
```

Eventualmente, ma ciò è da verificare, sarebbe opportuno attribuire anche il bit Sticky (*Save text image*), in modo da consentire la cancellazione solo al proprietario dei file contenuti:

```
# chmod o+trw /home/.samba/profiles [Invio]
```

Profili personali

In base alla configurazione proposta, si determina che i profili personali, definiti dalla direttiva '**`logon path`**', vengono collocati nel percorso '`\\`***`servente_smb`***`\profiles\`***`utente`***'.[1] In pratica, in questo modo si rimanda alla sezione '**`profiles`**', in cui si fa riferimento alla directory '`/home/.samba/profiles/`'. Pertanto, l'utente '**`tizio`**', si troverebbe ad avere i propri profili nella directory '`/home/.samba/profiles/tizio/`'.

Per garantire che tutto funzioni correttamente, il comando '**`nlnxrc user add`**' provvede a creare tale directory vuota, assegnandone la proprietà all'utente relativo. Nello stesso modo, la cancellazione di un utente con il comando '**`nlnxrc user del`**', provvede anche a cancellare tale directory.

Script di avvio

Nella configurazione proposta per Samba, appare la sezione '**`netlogon`**', con la quale si dichiara una directory, dove, eventualmente, si può collocare lo script da eseguire al collegamento di un utente. In tal caso, nella sezione '**`global`**' va aggiunta la direttiva '**`logon script`**':

```
[global]
    …
    logon script = logon.bat
    …
```

Secondo questo esempio, si tratta di predisporre il file '`logon.bat`' e collocarlo nella directory '`/home/.samba/netlogon/`'. Deve trattarsi di un file di testo, con i codici di interruzione di riga adatti al Dos. Eventualmente si può usare un programma come '**`unix2dos`**'.

Utenze per gli elaboratori MS-Windows

Gli elaboratori MS-Windows vanno aggiunti agli utenti Unix e Samba, utilizzando il nome predisposto per il protocollo NetBIOS. Nella figura dell'esempio si ipotizza di avere usato nomi del tipo '**`PC01`**', '**`PC02`**', '**`PC03`**',... Lì, in particolare, viene evidenziato l'elaboratore '**`PC07`**'.

Queste utenze particolari vanno aggiunte utilizzando soltanto lettere minuscole (pertanto, '**`PC07`**' diventa '**`pc07`**'), facendo in modo che nei file '`/etc/passwd`' e '`/etc/samba/smbpasswd`' appaiano conclusi con un dollaro (pertanto il nome '**`PC07`**' viene archiviato, in realtà, come '**`pc07$`**'). Questo lavoro viene svolto dallo script '**`nlnxrc`**', in modo da evitare errori:

```
# nlnxrc machine add [Invio]

  .--Add a new Win machine-----.
  | Please insert the new Win  |
  | machine name:              |
  | .------------------------. |
  | |                        | |
  | '------------------------' |
  |----------------------------|
  |   <  OK   >   <Cancel>     |
  '----------------------------'

pc07 OK

  .Full Win machine description--.
  | Please insert the machine    |
  | full description.            |
  | .--------------------------. |
  | |                          | |
  | '--------------------------' |
  |------------------------------|
  |   <  OK   >   <Cancel>       |
  '------------------------------'

Laboratorio informatica 3 OK
```

La creazione di queste utenze coincide con la creazione di altrettante directory vuote a partire dalla gerarchia '`/home/.samba/machines/`'. Tali directory non hanno alcuna utilità particolare, ma vengono prodotte ugualmente per esigenze che potrebbero manifestarsi in futuro.

Eventualmente, sempre con '**`nlnxrc`**' potrebbero essere eliminate tali utenze speciali:

```
# nlnxrc machine del pc07 [Invio]
```

Naturalmente, oltre a Mario Rossi, Antonio Bianchi e Giuseppe Verdi trovano la funzione di aggiunta di un elaboratore MS-Windows quando accedono al servente NLNX con la loro utenza speciale, tra le voci del menù prodotto dallo script '**`ADMIN9`**'.

Configurazione di MS-Windows XP Professional: disabilitazione della connessione cifrata

Perché MS-Windows XP possa essere associato a un dominio gestito da Samba, **potrebbe** essere necessario disabilitare alcune opzioni

relative all'uso della crittografia nella comunicazione per tale funzione. In pratica, ciò riguarda soltanto le versioni più vecchie di Samba e meno aggiornate di MS-Windows XP Professional; a ogni modo, in caso di difficoltà, si deve procedere attraverso le voci successive, ma in qualità di utente '**`Administrator`**', o equivalente:

- *Pannello di controllo*
  - *Prestazioni e manutenzione* (questa voce potrebbe essere saltata)
  - *Strumenti di amministrazione*
    - *Criteri di protezione locali*
      - *Criteri locali*
      - *Opzioni di protezione*
      - *Membro di dominio: ...*
        Membro di dominio: ...
        Membro di dominio: ...
        Membro di dominio: ...

Tutte le voci *Membro di dominio* vanno disattivate.

Figura u37.6. Pannello di controllo: selezione della voce *Strumenti di amministrazione.*

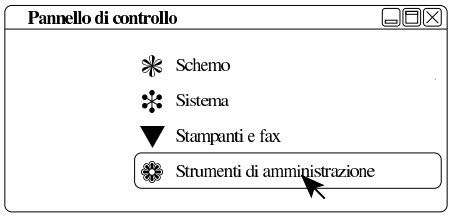


Figura u37.7. Strumenti di amministrazione: selezione della voce *Criteri di protezione locali.*

Figura u37.8. Impostazioni protezione locale: selezione della voce *Criteri locali.*

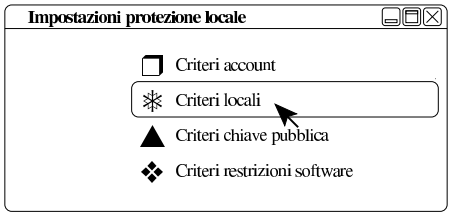


Figura u37.9. Opzioni di protezione: selezione della voce *Opzioni di protezione.*

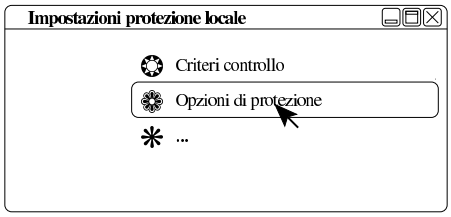


Figura u37.10. Impostazioni protezione locale: disattivazione delle voci *Membro di dominio.*

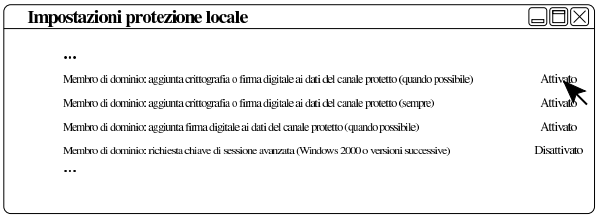
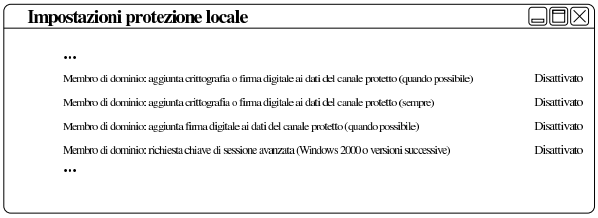


Figura u37.11. Impostazioni protezione locale: risultato della disattivazione delle voci *Membro di dominio.*



Configurazione di MS-Windows XP Professional: associazione al dominio

Inizialmente, MS-Windows XP si trova probabilmente a funzionare gestendo semplicemente i gruppi di lavoro. Per fare in modo di centralizzare le utenze occorre associarlo a un «dominio». Sulla base della configurazione proposta per Samba, il dominio in questione sarebbe denominato «NLNX», secondo la direttiva '**`workgroup`**':

```
[global]
    ...
    workgroup = NLNX
    ...
```

Presso l'elaboratore MS-Windows XP, con il nome '**`PC07`**', occorre procedere secondo i passi evidenziati dalle figure successive, ma occorre agire in qualità di utente '**`Administrator`**', o equivalente:

Figura u37.13. Pannello di controllo: selezione della voce *Sistema.*

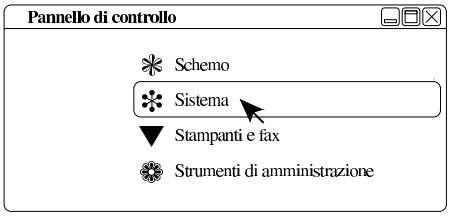


Figura u37.14. Proprietà del sistema: cambiamento del nome o dell'associazione a un dominio.

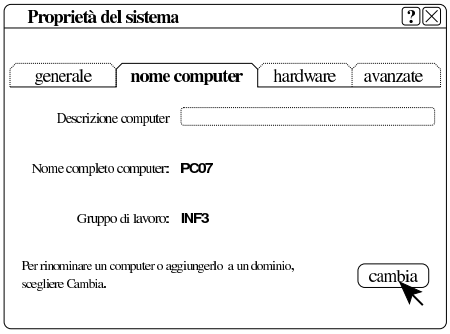
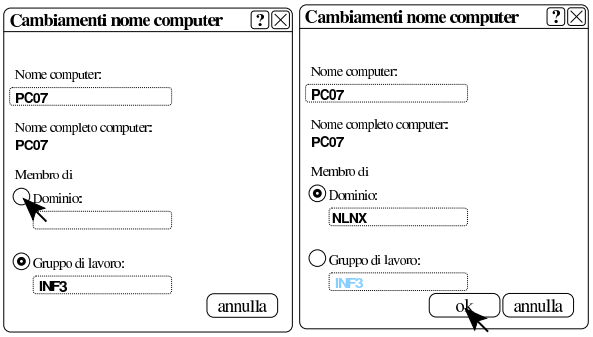
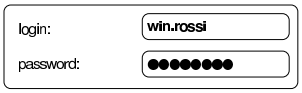


Figura u37.15. Cambiamenti nome computer: selezione del dominio e conferma.



Quando si vuole associare il dominio e confermare, viene richiesta l'indicazione di un'utenza «amministrativa», ovvero riconosciuta come tale presso il servente Samba. Ma attenzione: dal punto di vista di MS-Windows, queste utenze amministrative hanno il prefisso «win.»; pertanto, in base all'esempio, si tratta di '**`win.rossi`**', '**`win.bianchi`**' e '**`win.verdi`**'.

Figura u37.16. Richiesta di identificazione per l'utente amministrativo con cui ottenere l'aggiunta del dominio.



Al termine viene richiesto di riavviare il sistema per poter rendere operative le modifiche. Al riavvio può essere scelto se utilizzare le utenze locali preesistenti o il dominio appena collegato.

Configurazione di MS-Windows 7: definizione di due voci nel registro di sistema

«

Perché MS-Windows 7 possa essere associato a un dominio gestito da Samba si devono creare due voci nel «registro di sistema», ovvero in ciò che si gestisce attraverso il programma '**`regedit`**'.

Le voci da aggiungere vanno collocate nel percorso '`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\LanmanWorkstation\parameters\`':

```
Computer
  |
  |-HKEY_CLASSES_ROOT
  |-HKEY_CURRENT_USER
  |-HKEY_LOCAL_MACHINE
  |  |
  |  '->SYSTEM
  |     |
  |     '->CurrentControlSet->services->LanmanWorkstation->parameters
  |
  |-HKEY_USERS
  '-HKEY_CURRENT_CONFIG
```

Figura u37.18. Per avviare il programma '**`regedit`**' occorre digitarne il nome nel campo di ricerca, completando alla fine con [*Invio*].

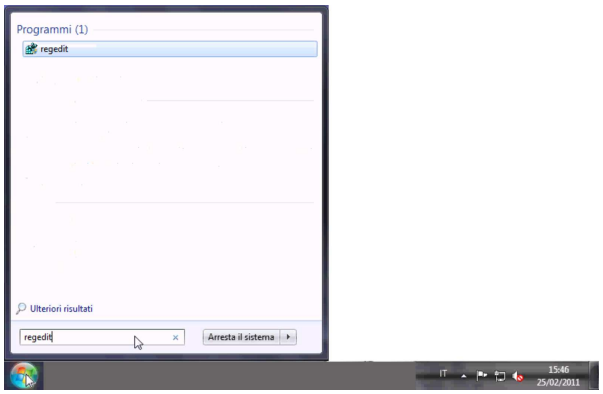
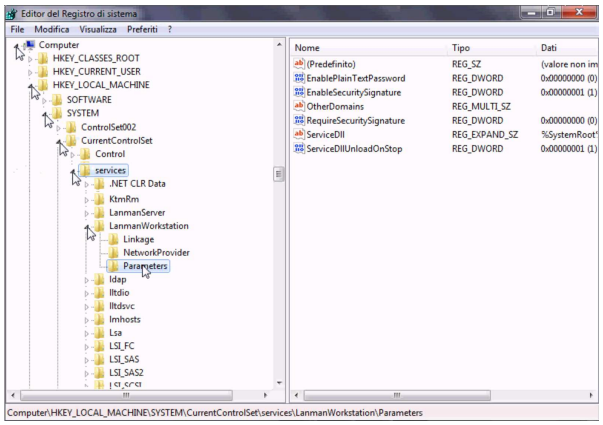


Figura u37.19. Svolgimento del percorso '`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\LanmanWorkstation\parameters\`'.



Le voci da aggiungere sono di tipo «DWORD» (nel senso di interi a 32 bit), denominate ***DomainCompatibilityMode*** e ***DNSNameResolutionRequired***. Alla prima di queste due voci si associa il valore 1, mentre alla seconda si deve lasciare il valore zero.

Figura u37.20. Creazione di una voce.

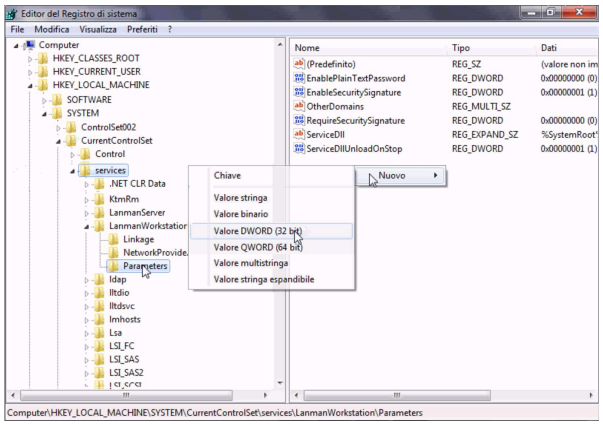


Figura u37.21. Creazione di una voce e modifica del suo contenuto.

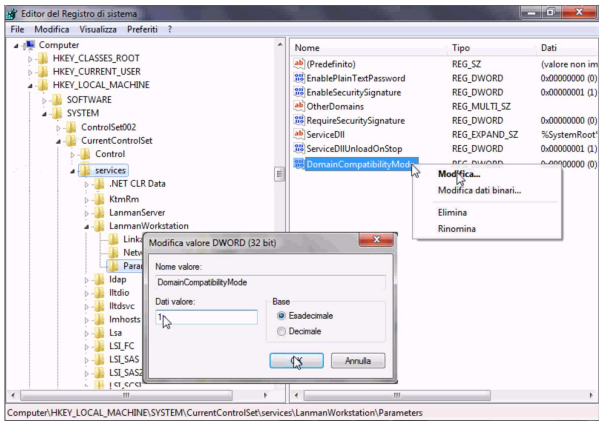
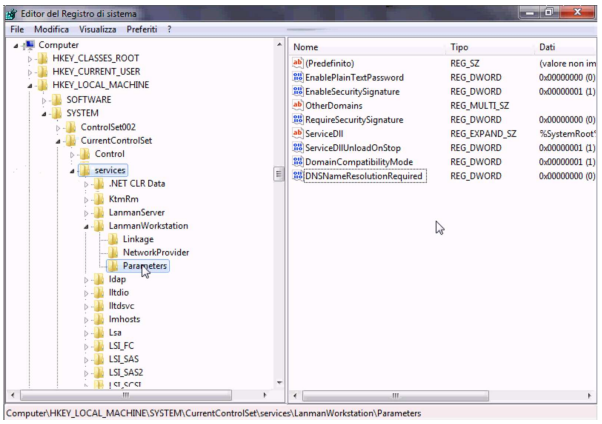


Figura u37.22. Dopo la creazione delle due voci.



Configurazione di MS-Windows 7: associazione al dominio

«

Inizialmente, MS-Windows 7 si trova probabilmente a funzionare gestendo semplicemente i gruppi di lavoro. Per fare in modo di centralizzare le utenze occorre associarlo a un «dominio». Sulla base della configurazione proposta per Samba, il dominio in questione sarebbe denominato «NLNX», secondo la direttiva '**`workgroup`**':

```
[global]
    ...
    workgroup = NLNX
    ...
```

Tuttavia, negli esempi seguenti si fa riferimento al dominio '**`INF`**' e il nome dell'elaboratore risulta essere '**`PC29L-VAIO`**'. Per prima cosa occorre ricordare di aggiungere la macchina '**`pc29l-vaio`**' alla gestione di Samba, usando solo lettere minuscole. Attraverso '**`nlnxrc`**' si procede con il comando seguente:

```
# nlnxrc machine add [Invio]
```

```
.--Add a new Win machine-----.
| Please insert the new Win  |
| machine name:              |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK   >   <Cancel>     |
'----------------------------'
```

`pc29l-vaio` OK

```
.Full Win machine description--.
| Please insert the machine    |
| full description.            |
| .--------------------------. |
| |                          | |
| '--------------------------' |
|------------------------------|
|   <  OK   >   <Cancel>       |
'------------------------------'
```

`Laboratorio informatica 5` OK

Quindi si può procedere con MS-Windows 7, selezionando la voce *Proprietà*, dal menù *Computer* (usando però il tasto destro del mouse).

Figura u37.26. Accesso alle proprietà.

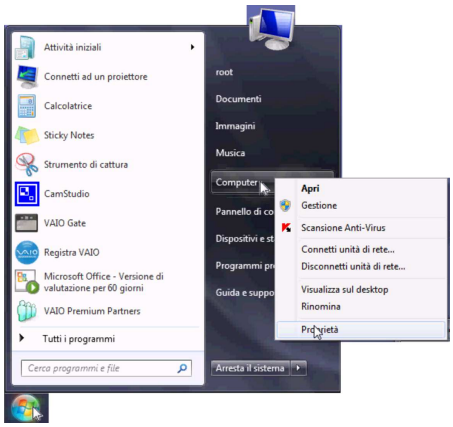


Figura u37.27. Selezione delle impostazioni avanzate.

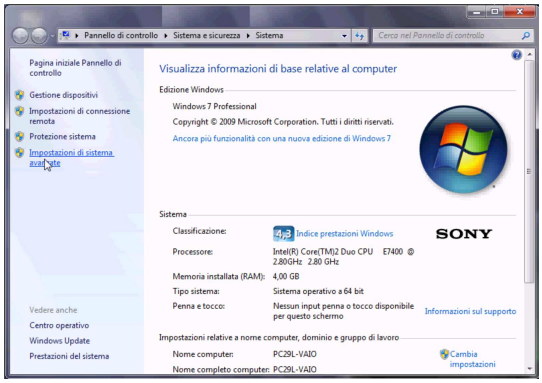


Figura u37.28. Nome dell'elaboratore.

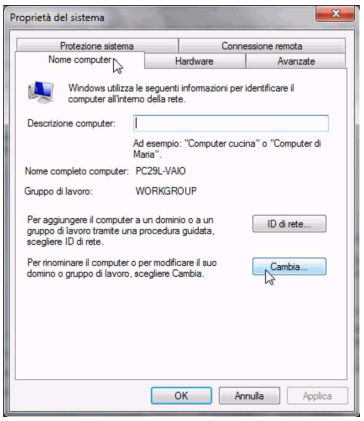
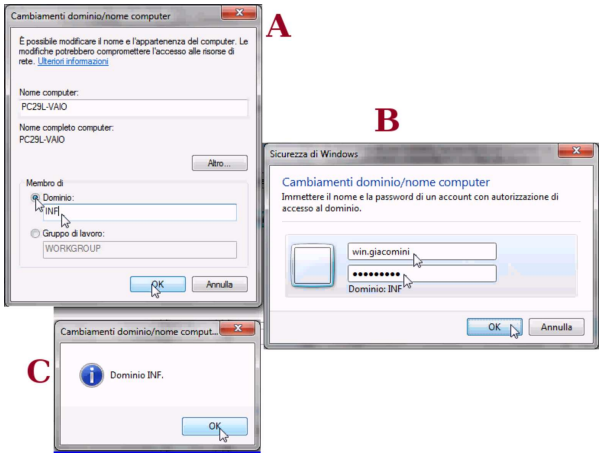


Figura u37.29. Associazione al dominio '**INF**' attraverso l'operato dell'amministratore '**win.giacomini**'.



L'utenza '**win.giacomini**' dell'esempio, fa parte di quelle di Samba e ha i privilegi amministrativi. Al termine, dopo la conferma, potrebbe apparire una segnalazione di errore, da ignorare.

Figura u37.30. Errore da ignorare al termine della procedura di associazione al dominio di Samba.

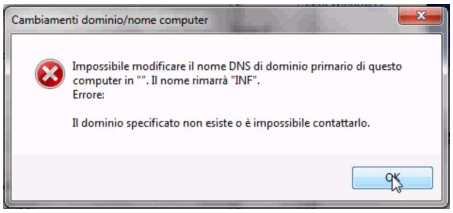
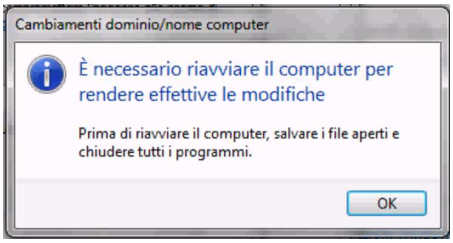


Figura u37.31. Al termine è necessario riavviare per mettere in pratica le modifiche.



Spegnimento del servente NLNX

Se l'elaboratore che svolge il ruolo di servente NLNX deve essere spento, è ragionevole attendersi che gli elaboratori MS-Windows, se usati durante tale inattività del servente NLNX, non siano in grado di accedere al dominio relativo. Tuttavia, per utilizzare gli elaboratori MS-Windows viene richiesto ugualmente il riconoscimento, almeno in qualità di utente locale.

Servente WINS

Nella configurazione di Samba, mostrata come esempio, appare la direttiva '**wins support = yes**', con la quale si ottiene di fornire anche il servizio WINS per gli elaboratori MS-Windows. Tali elaboratori, per potersene avvalere, devono essere configurati al riguardo; tuttavia, va considerato che in tal modo, il servente NLNX non può più essere spento, almeno fino a quando ci sono elaboratori che hanno bisogno di quel servizio.

Riferimenti

- Fulvio Ferroni, *Samba e OpenLDAP*
  http://linuxdidattica.org/docs/altre_scuole/planck/samba/
- By Jay Ts, Robert Eckstein, David Collier-Brown, *Using Samba, 2nd Edition*, 2003, O'Reilly & Associates, ISBN: 0-596-00256-4
  http://www.faqs.org/docs/samba/toc.html
- Jelmer R. Vernooij, John H. Terpstra, Gerald (Jerry) Carter, *The official Samba 3.2.x HOWTO and reference guide*, 2008
  http://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/
  http://us1.samba.org/samba/docs/Samba3-HOWTO.pdf

[1] La direttiva '**logon path**' riguarda MS-Windows NT/2000/XP.

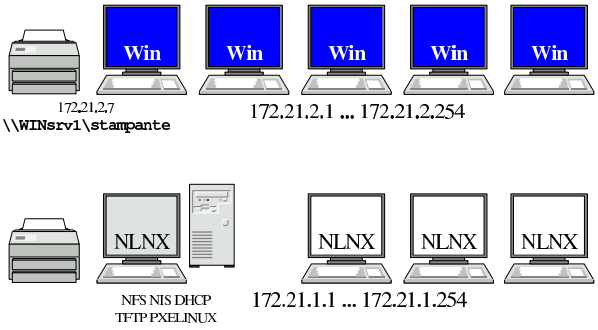
Installazione indolore di NLNX in una rete di elaboratori MS-Windows

Sequenza di avvio 179
File system principale attraverso la rete 179
Configurazione del DHCP 180
Configurazione dell'avvio 180
Riferimenti 181

Nelle sezioni successive si vuole dimostrare come si possa utilizzare NLNX in un contesto provvisto di una rete fisica adeguata, in cui sia possibile disporre di almeno un servente NLNX e ci siano molti altri elaboratori organizzati con sistemi operativi differenti (presumibilmente MS-Windows).

A titolo di esempio si propone inizialmente una situazione come quella descritta nella figura successiva, dove la rete fisica è unica e gli elaboratori sono divisi eventualmente in reti logiche. Il servente NLNX è l'unico che offra il servizio DHCP su tutta la rete fisica, inoltre, in condizioni normali, tale servente è utilizzato da un gruppo di elaboratori clienti NLNX. Naturalmente, il servente NLNX viene usato per tutti i servizi previsti da questa distribuzione; in particolare l'amministrazione delle utenze (NIS), la condivisione delle directory personali (cartelle personali) attraverso il protocollo NFS, SMB (Samba) ed eventualmente SSH, l'attribuzione automatica degli indirizzi IPv4 e l'indicazioni di altre informazioni con il DHCP, l'avvio remoto attraverso PXELINUX e il protocollo TFTP.

Figura u38.1. Situazione iniziale in cui gli elaboratori con indirizzi 172.21.1.* funzionano con NLNX e si avvalgono di un servente, il quale, tra l'altro, fornisce i servizi NFS, NIS, DHCP, TFTP e l'avvio remoto tramite PXELINUX.



Sequenza di avvio

Presso gli elaboratori che ospitano presumibilmente un sistema MS-Windows occorre riconfigurare il BIOS in modo da consentire l'avvio attraverso la rete (questa possibilità è ammissibile solo se l'interfaccia di rete è incorporata nella scheda madre), quindi occorre far sì che l'avvio dalla rete sia tentato prima delle altre possibilità. In tal modo, gli elaboratori in questione, all'accensione, andrebbero sempre a interpellare PXELINUX presso il servente NLNX.

Per NLNX, la configurazione standard di PXELINUX prevede che la voce di avvio predefinita consista nel richiamare l'avvio del disco fisso locale. Pertanto, se dopo alcuni secondi non si fa nulla, gli elaboratori configurati per avviarsi dalla rete si troverebbero ad avviare il sistema operativo locale, senza altre conseguenze.

Si osservi che in questo modo si evita di dover installare localmente, su tali elaboratori, un sistema di avvio che consenta di scegliere tra i vari sistemi che potrebbero convivere nel disco fisso locale. In altri termini, si evita di mettere a disagio il sistema operativo usato normalmente presso di loro.

File system principale attraverso la rete

Il modo meno problematico di usare NLNX consiste nell'avviarlo dalla rete, condividendo il file '`nlnx.img`' attraverso il protocollo NFS, ma in sola lettura, come descritto nella sezione u28. In pratica, presso il servente si colloca questo file nella directory '`/opt/nlnx/`' e ci si assicura che il protocollo NFS lo renda disponibile a tutti, in sola lettura, senza modificare i privilegi dell'utente con UID 0.

Configurazione del DHCP

Presso il servente NLNX va organizzato il servizio DHCP associando correttamente gli indirizzi fisici delle interfacce di rete agli indirizzi IPv4 utilizzati nella rete locale. Per esempio potrebbe trattarsi di un elenco simile a quello seguente:

```
…
172.21.1.1 00:D0:41:01:1C:F4
172.21.1.2 00:D0:41:01:1C:F5
172.21.1.3 00:D0:41:01:1C:F6
172.21.1.4 00:D0:41:01:1C:F7
…
172.21.2.1 00:D0:41:01:1B:F7
172.21.2.2 00:D0:41:01:1B:F8
172.21.2.3 00:D0:41:01:1B:F9
172.21.2.4 00:D0:41:01:1B:FA
…
```

Per scoprire gli indirizzi fisici degli elaboratori, quando questi sono in funzione è sufficiente eseguire un «ping» e osservare poi la tabella ottenuta dal comando '**`arp`**':

```
$ ping 172.21.2.1 [Invio]

...

[Ctrl c]

$ arp [Invio]

Address         HWtype  HWaddress           Flags Mask  Iface
…
172.21.2.1      ether   00:D0:41:01:1B:F7   C           eth0
…
```

Configurazione dell'avvio

Una volta predisposta la configurazione dell'avvio nel BIOS degli elaboratori ospitanti e dopo aver copiato i file-immagine di NLNX, conviene aggiungere nel servente NLNX una voce di avvio personalizzata. Si tratta di intervenire nel file '`/var/lib/tftpboot/pxelinux/pxelinux.cfg/default`'.

Gli elaboratori ospitanti potrebbero richiedere il caricamento esplicito di alcuni moduli, inoltre potrebbe essere necessario definire una configurazione particolare della grafica; infine, potrebbe essere il caso di indirizzare la stampa verso una stampante di rete più vicina rispetto a quella indicata automaticamente dal servizio DHCP del servente NLNX. Ecco un esempio di questa voce aggiuntiva, ipotizzando che la stampante in questione richieda un filtro adatto al tipo «laserjet»:

```
…
label guest
  kernel vmlinuz
  append n_boot=auto root=/dev/ram0 ro init=/linuxrc ↩
↪       initrd=nlnxrd.img ramdisk_size=30720 ↩
↪       n_setupdelay=8 ↩
↪       n_modules=atl1:r8169:ehci_hcd:uhci_hcd ↩
↪       n_xorg_conf=radeon,,,, ↩
↪       n_smb_prn_server=//winsrv1/stampante ↩
↪       n_lpr_filter=laserjet
…
```

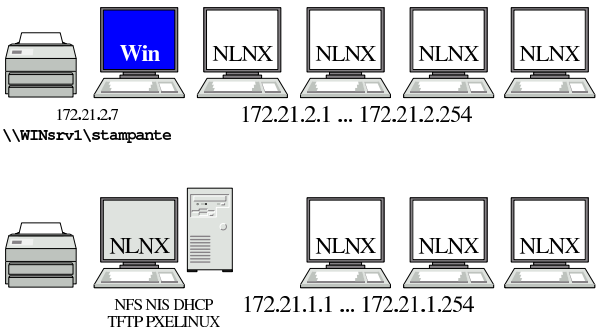
Lo stesso esempio, utilizzando direttamente l'indirizzo IPv4:

```
…
label guest
  kernel vmlinuz
  append n_boot=auto root=/dev/ram0 ro init=/linuxrc ↩
↪       initrd=nlnxrd.img ramdisk_size=30720 ↩
↪       n_setupdelay=8 ↩
↪       n_modules=atl1:r8169:ehci_hcd:uhci_hcd ↩
↪       n_xorg_conf=radeon,,,, ↩
↪       n_smb_prn_server=//172.21.2.7/stampante ↩
↪       n_lpr_filter=laserjet
…
```

In tal caso, per avviare il sistema NLNX ospitato andrebbe inserito il nome «guest» all'avvio.

Se invece si opta per l'installazione del file-immagine '`nlnx.img`' nel solo elaboratore servente, va sostituita l'opzione '**`n_boot=auto`**' con '**`n_boot=net`**'.

Figura u38.6. Situazione percepita durante il funzionamento di NLNX, avviato presso gli elaboratori che normalmente eseguono un sistema MS-Windows.



È bene osservare il problema delle utenze associato alla stampa verso una stampante condivisa: se l'elaboratore con il sistema MS-Windows che offre la condivisione è stato configurato con la gestione delle utenze personali, come descritto nella sezione u28, la stampa avviene solo se si fornisce un nominativo utente e una parola d'ordine valida. A questo proposito vanno usate le opzioni '**`n_smb_prn_user`**' e '**`n_smb_prn_passwd`**'. Supponendo che l'elaboratore 172.21.2.7 dell'esempio fornisca la stampante in condivisione in qualità di utente amministratore ('**`Administrator`**') e che tale utente non abbia alcuna parola d'ordine, la configurazione di avvio diventa:

```
…
label guest
  kernel vmlinuz
  append … n_smb_prn_server=//172.21.2.7/stampante ↩
↪       n_lpr_filter=laserjet↩
↪       n_smb_prn_user=root
…
```

Pertanto, si scrive '**`n_smb_prn_user=root`**' e si intende '**`Administrator`**'.

Riferimenti

- Fulvio Ferroni, *Samba e OpenLDAP*
  http://linuxdidattica.org/docs/altre_scuole/planck/samba/
- By Jay Ts, Robert Eckstein, David Collier-Brown, *Using Samba, 2nd Edition*, 2003, O'Reilly & Associates, ISBN: 0-596-00256-4
  http://www.faqs.org/docs/samba/toc.html
- Jelmer R. Vernooij, John H. Terpstra, Gerald (Jerry) Carter, *The official Samba 3.2.x HOWTO and reference guide*, 2008
  http://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/
  http://us1.samba.org/samba/docs/Samba3-HOWTO.pdf

Directory personale in un disco esterno 183
Utenze generiche 184
Messaggi di errore 184
Programmi «duri a morire» 184
Studenti troppo furbi 184
Vincolare gli utenti a un certo gruppo di postazioni 185

NLNX è organizzato secondo una struttura particolare. In questo capitolo si raccolgono osservazioni e suggerimenti che non hanno trovato spazio in altre sezioni relative a NLNX.

Directory personale in un disco esterno

Utilizzando NLNX avviato da un disco ottico *live*, può essere comodo gestire i propri dati personali utilizzando una memoria esterna, come un'unità USB a disco o allo stato solido. Per fare questo basta preoccuparsi di innestare il disco e probabilmente conviene decidere di usare uno degli utenti comuni stabiliti, sistemando di conseguenza la proprietà della directory radice dell'unità esterna:

```
# fdisk /dev/sda [Invio]

...
# mkfs.ext3 /dev/sda1 [Invio]

# mount /mnt/sda1 [Invio]

# chown tizio:tizio /mnt/sda1 [Invio]

# umount /mnt/sda1 [Invio]
```

I passaggi mostrati abbreviano e semplificano la procedura per creare una partizione (la prima) in un disco USB o in una memoria solida USB, facendo in modo che l'utente '**tizio**' la possa utilizzare come vuole.

Così facendo, utilizzando l'utenza '**tizio**' si possono salvare dati in questo disco, purché prima venga innestato. Tuttavia, rimanendo a questo livello di utilizzo, manca la possibilità di modificare in modo duraturo la configurazione personale dell'utente, perché i dati contenuti nella directory personale vanno perduti. Per risolvere questo problema, si può fare in modo di copiare la struttura iniziale della directory personale dell'utente '**tizio**' nel disco esterno, avendo cura ogni volta di innestarlo nel modo giusto:

```
# mount /mnt/sda1 [Invio]

# cp -dpRv /home/tizio /mnt/sda1 [Invio]

# umount /mnt/sda1 [Invio]
```

Eventualmente si può decidere di gestire in maniera diversa il disco esterno, per esempio inserendo il contenuto dei dati personali dell'utente in una sottodirectory, ma il procedimento non cambia; quando si vuole usare quella directory personale occorre prima agire come utente '**root**' seguendo lo schema seguente:

```
# mount /mnt/sda1 [Invio]

# mount --bind /mnt/sda1 /home/tizio [Invio]
```

Quindi ci si può identificare come utente '**tizio**'. Se la directory personale dell'utente si trovasse invece, per esempio, nella sottodirectory '`mia/directory/personale/`' nel disco esterno, basterebbe cambiare il collegamento finale nel modo seguente:

```
# mount --bind /mnt/sda1/mia/directory/personale ↩
↪       /home/tizio [Invio]
```

Utenze generiche

Quando si installa NLNX in modo che utilizzi il file system in lettura e scrittura, le utenze generiche costituite dai nomi '**tizio**', '**caio**' e gli altri, vengono disabilitate, nel senso che viene tolta la parola d'ordine, ma per il resto sono intatte.

Naturalmente ci possono essere situazioni in cui gli utenti generici predefiniti sono totalmente inutili. In questi casi conviene provvedere manualmente alla loro eliminazione, soprattutto per ciò che riguarda le directory personali. Tuttavia, se si tratta di una copia di NLNX installata per poter generare successivamente un DVD «live», conviene lasciare le voci di questi utenti nei file '`/etc/passwd`' e '`/etc/group`', per evitare che l'installazione di un programma che richiede la creazione di un utente fittizio vada a utilizzare proprio i numeri UID e GID che sono previsti invece per gli utenti '**tizio**' e gli altri. Dovendo mantenere questi utenti nei file '`/etc/passwd`' e '`/etc/group`' si può assegnare al posto della shell comune il file '`/bin/false`'.

Se si lasciano le directory personali degli utenti generici predefiniti, in un elaboratore che offre le directory personali attraverso il protocollo NFS, se questo servizio viene utilizzato tramite DVD, dal momento che lì tali utenze sono attive, è possibile salvare i file attraverso NFS. Si osservi che è sufficiente il protocollo NFS, perché le utenze generiche predefinite hanno numeri UID al di sotto del livello minimo previsto per la gestione attraverso il NIS (in base alla configurazione che prevede come UID minimo il numero 1000). Se questo avviene in un laboratorio didattico, significa che gli studenti possono scambiare file tra di loro, perché, per esempio, chiunque può diventare '**tizio**'.

Messaggi di errore

Durante il funzionamento in modalità grafica, viene mostrato normalmente un riquadro contenente i messaggi generati dai programmi. Questi messaggi comprendono quanto emesso dallo standard output e dallo standard error. Tra i messaggi appaiono anche quelli generati dal gestore di finestre e in particolare sarebbe molto frequente il messaggio seguente:

```
[FVWM][get_menu_options]: <<ERROR>> invalid rectangle ↩
↪geometry
```

Questa segnalazione in particolare sembra essere generata erroneamente, per un problema che in realtà non sussiste. Dal momento che questo messaggio in particolare verrebbe emesso ogni volta che si apre il menù, dando così un fastidio inutile, il comando che mostra il riquadro dei messaggi fa in modo che non appaia.

Programmi «duri a morire»

Durante il funzionamento in modalità grafica, può capitare di avviare dei programmi che poi, non vogliono saperne di terminare il loro funzionamento quando la sessione grafica viene conclusa. Questo tipo di inconveniente si è manifestato, in modo particolare, con il programma '**alsamixergui**', che per questa ragione non viene più installato. Per lo stesso motivo, non sono disponibili funzioni per generare sfondi dinamici.

Studenti troppo furbi

NLNX è organizzato in modo particolare per l'uso in un laboratorio didattico, per studenti della scuola media superiore. Di norma, un esercizio o un compito richiede che ognuno lavori utilizzando i propri dati, senza poter accedere alle directory personali degli altri utenti (soprattutto in considerazione del fatto che si presume i dati personali siano centralizzati e condivisi nella rete locale).

Per rendere più complicata la condivisione indesiderata dei dati, alla creazione delle utenze, la proprietà della directory personale (solo la directory, non il contenuto) viene attribuita a '**root**', lasciando il gruppo associato correttamente all'utente per il quale esiste tale

directory. Ciò comporta che l'utente non possa cambiare i permessi stabiliti per tale directory. Naturalmente, gli utenti hanno un proprio gruppo privati e al proprio gruppo sono concesse tutte le operazioni sulla directory. Ma oltre a questo viene attivato il bit Sticky (*Save text image*), in modo che possano essere rimossi solo i file che appartengono a chi chiede di rimuoverli.

I permessi iniziali consentono anche agli altri utenti di accedere (il permesso di «esecuzione», ovvero di attraversamento), allo scopo di rendere possibile l'interscambio di file tra studenti e docenti, oppure per consentire la pubblicazione di file nella directory '`~/public_html/`'. Eventualmente, per impedire che gli studenti possano pubblicare dei file utilizzando la directory '`~/public_html/`', questa può essere creata in modo che appartenga all'amministratore, togliendo tutti i permessi di accesso e di lettura: gli studenti non possono così cancellarla e ricrearla con permessi differenti.

Tuttavia, dal momento che, se gli studenti conoscono i nomi dei file possono ugualmente condividerli, si può applicare una politica più rigida, togliendo anche il permesso di accesso alla directory personale. A titolo di esempio, supponendo di voler limitare gli accessi a un'ipotetica classe 5A dell'anno scolastico 2012/2013, ammesso di avere organizzato correttamente la struttura delle directory personali, si potrebbe procedere così:

```
# cd /home/5A1213 [Invio]

# for s in * ; do chmod 1770 ; done [Invio]
```

In tal caso, però, lo script fornito come esempio per facilitare lo scambio di dati tra studenti e docenti non funzionerebbe più e andrebbe riscritto copiando materialmente i file.

Vincolare gli utenti a un certo gruppo di postazioni

Se gli elaboratori a cui gli utenti possono accedere sono tutti dotati di un sistema NLNX, è possibile imporre l'uso di certe postazioni, rispetto ad altre, ad alcuni utenti. Questa funzionalità non è guidata attraverso lo script '**nlnxrc**' e richiede un po' di lavoro.

Ogni elaboratore a cui si vuole sottoporre questo controllo deve contenere il file di testo '`/etc/nlnx/TTY_LOGIN_AT`', con l'elenco degli utenti da limitare, dove sono ammesse solo direttive che si compongono secondo il modello seguente:

utente indirizzo_ipv4 [*indirizzo_ipv4*]

In pratica, prima si mette il nome dell'utente, quindi, separandoli con degli spazi, gli indirizzi da cui si possono connettere. Si osservi l'esempio seguente:

```
tizio 172.17.1.23 172.17.1.25
caio 172.17.1.15
```

In questo caso, ciò che è scritto nel file indica che l'utente '**tizio**' è ammesso ad accedere da 172.17.1.23, oppure 172.17.1.25, mentre l'utente '**caio**' solo da 172.17.1.15. Perché il controllo sia efficace, è necessario che questo file sia copiato tale e quale in tutti gli elaboratori. Tuttavia, quando il sistema viene avviato dalla rete, ciò non è possibile, pertanto in quel caso, il file può essere collocato nella directory '`/opt/nlnx/configuration/default/`'

Gerarchia doppia 187
Creazione di una nuova versione 188
Configurazione predefinita di NLNX 188
Aggiornamento dei pacchetti installati 189
Kernel 189

Una volta installato NLNX, è possibile aggiungere o eliminare dei pacchetti applicativi secondo la procedura prevista dalla distribuzione GNU/Linux Debian. In condizioni normali, è sufficiente il comando '**`nlnxrc nlnx make`**' per riprodurre il tutto in un nuovo DVD.

Alle volte potrebbe essere necessario un adattamento più consistente e a tale scopo possono servire le note di questo capitolo.

Gerarchia doppia

Quando NLNX è installato nel disco fisso secondo la procedura normale, oltre alla struttura comune di file e directory, appare la directory '`/RO-FS/`', all'interno della quale, tra le altre cose, appare anche la directory '`RW-FS/`' (in pratica '`/RO-FS/RW-FS/`').

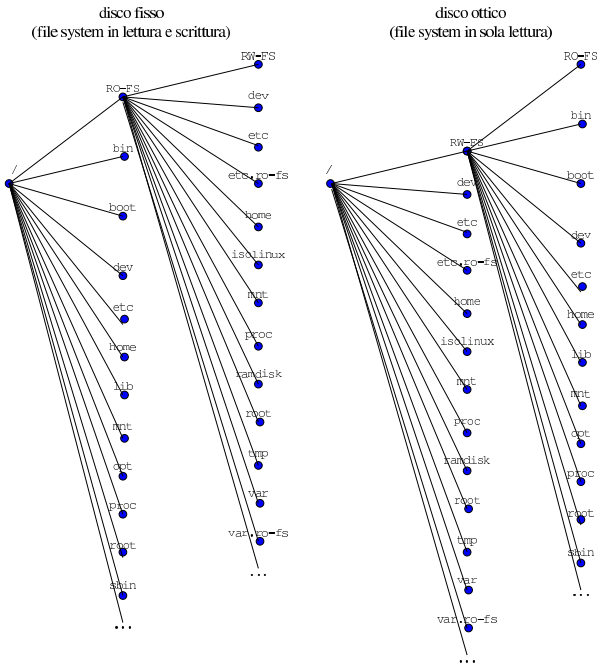
La directory '`/RO-FS/`' rappresenta la radice del DVD *live* che si va a creare, mentre la directory '`/RO-FS/RW-FS/`' deve innestare nuovamente la struttura installata nel disco fisso; pertanto, nel file '`/etc/fstab`' devono apparire due righe simili a quelle seguenti:

```
/dev/hda2 /            auto defaults,errors=remount-ro 0 1
/dev/hda2 /RO-FS/RW-FS auto defaults,noauto            0 0
```

In questo caso la seconda riga non prevede un innesto automatico all'avvio, ma a ciò provvede comunque '**`nlnxrc nlnx make`**'.

Quando si avvia il DVD *live*, la struttura che si vede si compone in particolare della directory '`/RW-FS/`', la quale contiene dati non modificabili e serve per copiare il suo contenuto nel disco fisso, quando si va a installare NLNX.[1]

Figura u40.2. Confronto tra i file system durante il funzionamento da disco fisso rispetto al funzionamento da disco ottico.



Creazione di una nuova versione

La creazione di un nuovo sistema NLNX si ottiene tramite il comando '**nlnxrc nlnx make**'. Lo script '**nlnxrc**' ('/etc/script/nlnxrc') è molto lungo e articolato; le istruzioni che riguardano la creazione di un nuovo disco sono racchiuse in una struttura '**if**'...'**elif**'...'**fi**'. In questa struttura si verifica se il comando selezionato è ammissibile, in base al fatto che l'avvio dello script avvenga da disco fisso o da DVD-ROM e al tipo di utente che lo esegue. L'istruzione che introduce la creazione di un nuovo disco è quella seguente:

```
elif [ "$UID" = "0" ] && [ "$CDROM" = "0" ] ↩
↪&& [ "$COMMAND" = "nlnx" ] && [ "$DIRECTIVE" = "make" ]
then
    istruzioni_creazione_disco
elif …
…
```

Tra le tante cose che avvengono qui dentro, si preparano le directory '/RO-FS/etc/' e '/RO-FS/var/', utilizzando anche il contenuto di '/RO-FS/etc.ro-fs/' e '/RO-FS/var.ro-fs/'.

In pratica, la directory '/RO-FS/etc/' viene ottenuta copiando la stessa struttura contenuta in '/etc/', sostituendo ogni file che non sia una directory con un collegamento simbolico che punta alla stessa cosa a partire da 'RW-FS/etc/'; in modo analogo procede la preparazione della directory '/RO-FS/var/'. Quindi, viene ricopiato sopra il contenuto di '/RO-FS/etc/' e di '/RO-FS/var/', quanto si trova dentro '/RO-FS/etc.ro-fs/' e '/RO-FS/var.ro-fs/' rispettivamente. Naturalmente, questa è una semplificazione; in pratica, prima di passare alla sovrapposizione delle directory '/RO-FS/*.RO-FS/', lo script fa qualche ritocco indispensabile.

Dovendo intervenire in modo da modificare il contenuto delle directory '/etc/' o '/var/' durante il funzionamento da disco ottico, si può agire nelle directory '/RO-FS/etc.ro-fs/' e '/RO-FS/var.ro-fs/', a meno che si tratti di una situazione che non si risolve con la semplice copia di qualcosa di diverso. Per esempio, se un collegamento simbolico contenuto in una di queste directory può creare problemi, lo si può sostituire con un file vero e proprio, che in fase di funzionamento da DVD risulterebbe modificabile. Quello che non si può fare è rimpiazzare una directory con un file e nemmeno cancellare qualcosa; in queste situazioni estreme, diventa necessario intervenire nello script '**nlnxrc**'.

Configurazione predefinita di NLNX

Lo script '/etc/init.d/nlnx.config' di NLNX rimpiazza alcuni file di configurazione con altri che hanno generalmente l'estensione '.nlnx'. La tabella successiva riepiloga alcuni di questi file.

Tabella u40.4. Alcuni file di configurazione predefiniti di NLNX, secondo lo script '/etc/init.d/nlnx.config'.

File originale	Configurazione predefinita	Descrizione
'/etc/oops/oops.cfg'	'/etc/oops/oops.cfg.nlnx'	Configurazione del servizio proxy HTTP.
'/etc/X11/fvwm/system.fvwm2rc'	'/etc/X11/fvwm/system.fvwm2rc.nlnx'	Si tratta della configurazione predefinita del menù di Fvwm, per NLNX.
'/etc/X11/xinit/xinitrc'	'/etc/X11/xinit/xinitrc.nlnx'	Si tratta della configurazione predefinita dello script usato per avviare il gestore di finestre con qualcosa sullo sfondo.
'/etc/X11/xorg.vesa'	'/etc/X11/xorg.vesa.nlnx'	Configurazione di X predefinita, per un adattatore standard VESA.
'/etc/X11/xdm/Xresources'	'/etc/X11/xdm/Xresources.nlnx'	Configurazione di XDM per l'accesso alla sessione grafica.
'/etc/xfe/xferc'	'/etc/xfe/xferc.nlnx'	Configurazione di XFE.

Molti altri file con estensione '.nlnx' servono a conservare una copia della configurazione standard di NLNX, ma non vengono rimpiazzati automaticamente.

Aggiornamento dei pacchetti installati

Una volta installato NLNX, l'aggiornamento o l'aggiunta di pacchetti può avvenire con gli strumenti consueti della distribuzione GNU/Linux Debian. In pratica conviene usare '**apt-get**', come spiegato nella sezione 7.7.

Dal momento che si presume NLNX venga installato prevalentemente a scuola, è da considerare che la rete tipica che si incontra in quel contesto obbliga l'attraversamento di un proxy HTTP, che spesso è in grado di consentire l'accesso esclusivamente da navigatori funzionanti su macchine MS-Windows. Teoricamente, '**apt-get**' può essere configurato per attraversare un proxy, come si può anche vedere negli esempi che appaiono nel file '/usr/share/doc/apt/examples/configure-index.gz', ma non è detto che il proxy della propria realtà consenta effettivamente questo approccio.

Per risolvere il problema occorre procurarsi i pacchetti in modi differenti, per esempio usando una macchina (MS-Windows) che consenta l'attraversamento, per accedere direttamente al sito *http://www.debian.org/* , da dove si può raggiungere la pagina di ricerca dei pacchetti (*http://www.debian.org/distrib/packages*). Una volta prelevati i pacchetti e copiati in qualche modo nell'elaboratore che si vuole aggiornare, si può tentare di usare '**dpkg**' con l'opzione '**-i**':

```
# dpkg -i file_deb… [ Invio ]
```

Purtroppo si tratta di un metodo brutale di installazione che rischia di bloccarsi per colpa di dipendenze che non sono soddisfatte; ma dagli errori che si ottengono si possono determinare quali pacchetti si devono ancora prelevare.

Nella sezione 7.11 vengono descritti vari accorgimenti per la gestione dei pacchetti Debian; in particolare viene mostrato un metodo per realizzare in proprio la struttura di una distribuzione (attraverso lo script '**make-packages**'), in modo da poter usare poi '**apt-get**' localmente. Anche con l'uso di '**apt-get**' rimane il problema delle dipendenze non soddisfatte, ma in questo modo si riesce almeno a realizzare qualcosa di generalizzato, che può risiedere facilmente in un disco esterno USB, o anche in un DVD masterizzato, per aggiornare facilmente le macchine che si vogliono usare.

Kernel

Il kernel di NLNX è molto simile a quello standard della distribuzione Debian, con la differenza più evidente che mancano del tutto le funzionalità ritenute inutili e ci sono più componenti incorporate direttamente nella parte principale (nel senso che non fanno parte di moduli separati). Vengono annotate nella tabella successiva alcune scelte fatte nella configurazione e le motivazioni relative.

Modulo	Voce di configurazione	Descrizione
`md_mod`	`CONFIG_MD=y`	Solo se questa funzione è incorporata nel file principale del kernel è possibile ottenere la scansione automatica degli insiemi RAID di dischi, con il comando '**`mdadm --auto-detect`**'.
`cramfs`	`CONFIG_CRAMFS=y`	Il file system del disco RAM iniziale è di tipo Cramfs e in quella fase la sua gestione non può trovarsi separata in un modulo.
	`CONFIG_USB_SUSPEND=n`	Utilizzando uno scanner USB Canon LiDE, avendo attivata la funzione '**`CONFIG_USB_SUSPEND`**', si ottengono solo scansioni completamente nere.

[1] Va ricordato che la struttura del file system del sistema avviato da un disco ottico *live* è la stessa per qualunque altro contesto in cui il file system va usato in sola lettura.

Accensione e spegnimento 192
Procedura per l'aggiunta di un utente 192
Controllo dei servizi 196
Controllo del numero massimo di pagine stampabili per volta 197
Utilizzo di elaboratori estranei al laboratorio 197

Oltre al responsabile del laboratorio, altri potrebbero eseguire alcune operazioni legate all'amministrazione tecnica dello stesso, per evitare di dover dipendere da una sola persona per ogni cosa. Queste persone dispongono di un'utenza amministrativa personale, presso l'elaboratore con indirizzo 172.17.1.254 (localmente o a distanza), il quale può essere raggiunto anche con l'indirizzo 192.168.0.71. Attraverso tale utenza amministrativa si ottiene un menù di funzioni prestabilito, come si vede nella figura seguente, che può contenere più voci se l'accesso avviene presso la console dell'elaboratore principale:

```
.----------------------Admin menu-------------------------.
| Admin limited menu                                      |
| .-------------------------------------------------------. |
| |   adduser         Add a new user                    | |
| |   passwd          Change a user's password          | |
| |   description     Change a user's description       | |
| |   user info       Show user info                    | |
| |   home info       Show home directory info          | |
| |   quota report    Show sorted user's quota          | |
| |   nis-make        Rebuild NIS database              | |
| |   add machine     Add a new MS-Windows computer     | |
| |   proxy access    HTTP proxy access permissions     | |
| |   admin passwd    Change your password              | |
| |   custom1         Custom script 1                   | |
| |   custom2         Custom script 2                   | |
| |   custom3         Custom script 3                   | |
| |   exit            Quit                              | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >         <Cancel>                  |
'-----------------------------------------------------------'
```

Attraverso questo menù è possibile, in particolare, aggiungere un'utenza, cambiare la parola d'ordine di un utente che ne fa richiesta e verificare la configurazione delle utenze.

Tabella u41.2. Descrizione delle funzioni principali disponibili alle utenze amministrative. Si osservi che la voce '**`lprm`**' è accessibile solo localmente, attraverso la console dell'elaboratore 172.17.1.254.

Funzione	Descrizione
`adduser`	Consente di aggiungere un'utenza al sistema.
`passwd`	Consente di cambiare la parola d'ordine associata a un'utenza.
`printer access`	Controlla gli accessi al servizio di stampa.
`user info` `home info`	Consente di avere informazioni su un'utenza.
`quota report`	Mostra l'elenco delle quote degli utenti, ordinata in modo decrescente per quantità di spazio utilizzato.
`proxy access`	Controlla gli accessi al servizio proxy HTTP.
`admin passwd`	Consente all'amministratore di cambiare la propria parola d'ordine necessaria per accedere.
`custom1`	Allinea le directory '`verifiche/`' e '`strumenti/`' tra docenti e studenti.
`custom2` `custom3`	Consente di controllare l'accesso alle unità di memorizzazione esterne.

Funzione	Descrizione
`custom3` `custom3`	Consente di controllare l'accesso a risorse esterne, tramite il proxy HTTP, nel proprio laboratorio o in altri, se configurati in modo appropriato.
`exit`	Esce dal menù.

Accensione e spegnimento

L'elaboratore con indirizzo 172.17.1.254 deve rimanere accesso sempre, anche durante la notte, perché svolge servizi necessari a tutta la rete locale (non solo per il laboratorio) e perché durante la notte può eseguire delle elaborazioni (per esempio la ricerca antivirus) che altrimenti appesantirebbero inutilmente le altre attività.

Per questa ragione, nel quadro elettrico va lasciato inserito l'interruttore generale e l'interruttore periferico (contrassegnato appositamente) che controlla l'alimentazione dell'elaboratore 172.17.1.254.

Gli elaboratori destinati agli utenti, vanno accesi e spenti all'occorrenza, ma in ogni caso è previsto un piano di spegnimento automatico giornaliero, per maggiore sicurezza.

Procedura per l'aggiunta di un utente

Dal momento che si prevede la presenza simultanea di un gran numero di utenze, si richiede a chi interviene per aggiungerne di nuove di farlo con un certo ordine.

Per prima cosa **si conviene che il nominativo scelto dall'utente cominci con il cognome** e continui, possibilmente, con il nome (per esempio «rossimario»), tenendo conto che si possono usare al massimo 15 caratteri alfabetici e numerici (lettere dalla «a» alla «z», minuscole, cifre da zero a nove).

Una volta inserito il nominativo, è necessario stabilire una sigla che viene usata per classificare l'utenza. Nel caso degli studenti che usano comunemente il laboratorio, va usata la sigla della classe (compresa la sezione e una lettera per distinguere il corso di studi) seguita dall'anno scolastico. La tabella seguente riepiloga alcuni esempi.

Tabella u41.3. Esempi di sigle da usare per raggruppare le utenze secondo il contesto per il quale vengono create.

Esempio	Contesto
`5A1213`	Utenze annuali, riferite agli alunni di 5A, create nell'anno scolastico 2012/2013, dove non è necessario specificare il corso di studi.
`4AS1213`	Utenze annuali, riferite agli alunni di 4A, corso «S», create nell'anno scolastico 2012/2013.
`DOCENTI0`	Utenze riferite ai docenti, da conservare attraverso gli anni scolastici.
`ITP00000`	Utenze riferite agli insegnanti tecnico-pratici, da conservare attraverso gli anni scolastici.
`TECNICI0`	Utenze riferita ai tecnici, da conservare attraverso gli anni scolastici.
`COLLSCOL`	Utenze riferite ai tecnici, da conservare attraverso gli anni scolastici.
`SEGRETER`	Utenze riferite al personale di segreteria, da conservare attraverso gli anni scolastici.
`DSGA0000`	Utenza riferita al direttore dei servizi generali e amministrativi, da conservare attraverso gli anni scolastici.
`DIRIGENT`	Utenza riferita al dirigente scolastico, da conservare attraverso gli anni scolastici.
`ESTERNI0`	Utenti che non fanno parte dell'istituto, ma che sono ammessi ad accedere alla rete didattica per qualche motivo.

Quando viene richiesto di inserire un utente, il programma che si occupa di questo consente di aggiungere dei dati ulteriori all'interno di un campo aggiuntivo. Ciò va usato per indicare i dati significativi dell'utenza, in base al contesto di utilizzo. Seguono due esempi; il primo riferito all'utente Mario Rossi che è uno studente di 5A «igea», il secondo riferito al professor Sempronio Dicembrino che è

docente di economia aziendale:

```
.----------------------Add a new user-----------------------.
| [en] Please insert the new user name (only lower case    |
| letters and numbers, minimum 8 and max 15 characters,    |
| the first character must be a letter).                   |
|                                                          |
| [it] Inserire il nominativo utente, composto             |
| preferibilmente da cognome e nome attaccati, ed          |
| eventualmente l'anno di nascita, per esempio             |
| "rossimario1990" (si possono usare solo lettere          |
| minuscole e cifre numeriche per un minimo di 8 e fino a  |
| un massimo di 15 caratteri, ma il primo carattere deve   |
| essere una lettera).                                     |
| |........9012345|                                        |
| .--------------------------------------------------.     |
| |                                                  |     |
| '--------------------------------------------------'     |
|----------------------------------------------------------|
|                 <  OK  >            <Cancel>             |
'----------------------------------------------------------'
```

`rossimario` OK

```
.--------------------Classify user-----------------------.
| [en] Please select a hierarchy name for the user: it   |
| will be used as an intermediate directory after        |
| "/home/".                                              |
|                                                        |
| [it] Inserire la classificazione dell'utente: viene    |
| usata come directory intermedia dopo "/home/".         |
| .--------------------------------------------------. |
| |              1AI1213   /home/1AI1213               | |
| |              1AM1213   /home/1AM1213               | |
| |              1AS1213   /home/1AS1213               | |
| |              1BI1213   /home/1BI1213               | |
| |              1BM1213   /home/1BM1213               | |
| |              1BS1213   /home/1BS1213               | |
| |              1CI1213   /home/1CI1213               | |
| |              1CM1213   /home/1CM1213               | |
| |              1DI1213   /home/1DI1213               | |
| |              DOCENTI0  /home/DOCENTI0/             | |
| |              --------  ----------------------     | |
| |              new       add a new one              | |
| '--------------v(+)---------------------------------' |
|                                                        |
|                                                        |
|--------------------------------------------------------|
|              <  OK  >        <Cancel>                 |
'--------------------------------------------------------'
```

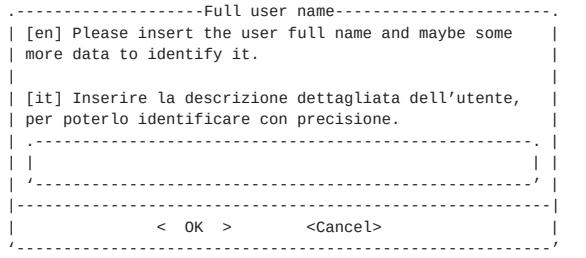
Non essendo già stata prevista la classificazione '**`5AI1213`**', si seleziona la voce '**`new`**' per passare all'inserimento manuale:

```
.--------------------Classify user-----------------------.
| [en] Please insert a hierarchy name for the user: it     |
| will be used as an intermediate directory after "/home/".|
| Please insert only upper case letters and digits (A-Z,   |
| 0-9; min 4, max 8 characters).                           |
|                                                          |
| [it] Inserire la classificazione dell'utente: viene      |
| usata come directory intermedia dopo "/home/".           |
| Inserire solo lettere maiuscole e cifre numeriche (A-Z,  |
| 0-9; min 4, max 8 caratteri).                            |
| |....5678|                                               |
| .--------------------------------------------------. |
| |2012                                              | |
| '--------------------------------------------------' |
|                                                          |
|----------------------------------------------------------|
|              <  OK  >            <Cancel>                |
'----------------------------------------------------------'
```

Viene cancellato il valore predefinito e viene assegnata la sigla '**`5AI1213`**':

[*Canc*][*Canc*][*Canc*][*Canc*]

`5AI1213` OK

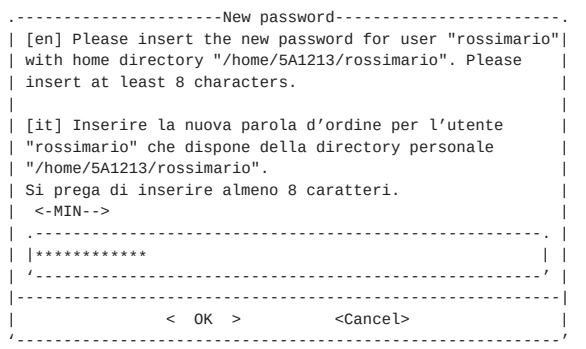


Vengono richiesti alcuni dati addizionali, che è bene compilare per poter individuare correttamente l'utente:

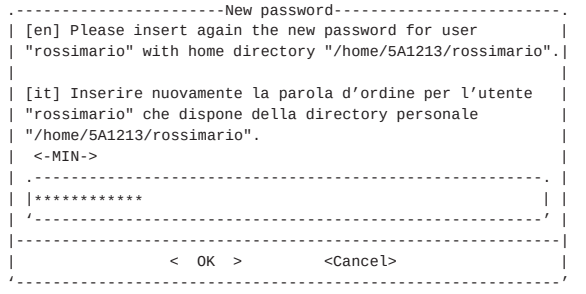
studente Rossi Mario 5A igea 2012/2013 [*Invio*]

```
Adding user 'rossimario'...
Adding new group 'rossimario' (1000).
Adding new user 'rossimario' (1000) with group 'rossimario'.
Creating home directory '/home/5AI1213/rossimario'.
Copying files from '/etc/skel'
```

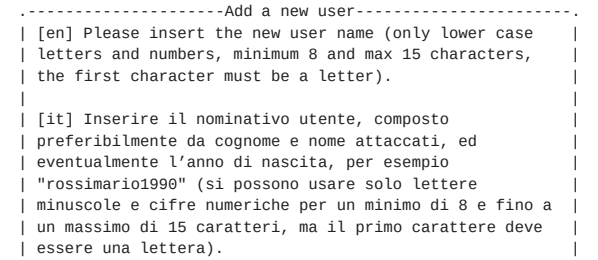
Al termine viene richiesto di inserire per due volte la parola d'ordine, cosa che deve essere fatta direttamente dalla persona per la quale si crea l'utenza.



L'inserimento corrisponde alla visualizzazione di una serie di asterischi.

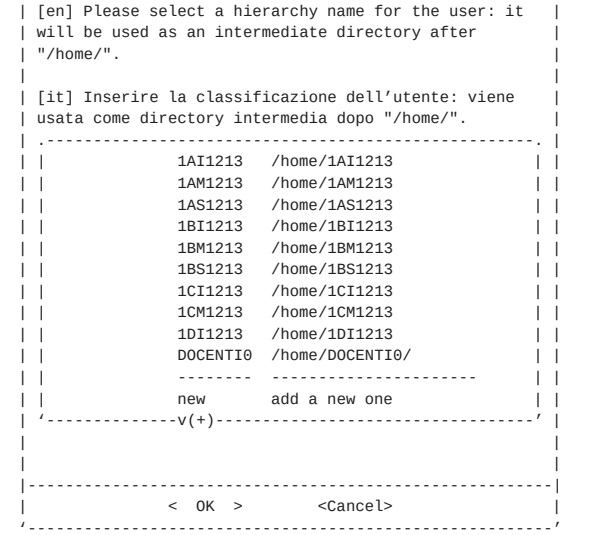


Dopo l'inserimento, per due volte, della parola d'ordine, se tutto è stato fatto senza errori, soprattutto se la parola d'ordine è stata inserita correttamente, l'operazione è completa e si può procedere con l'utente successivo:

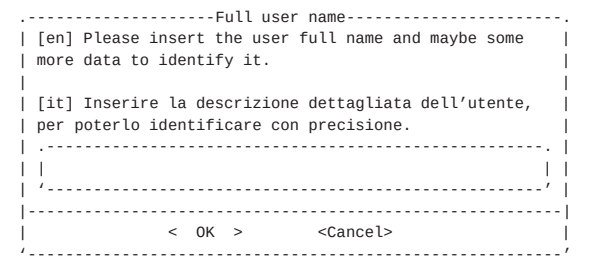


Passando all'inserimento dell'utente «Sempronio Dicembrino», la somma di cognome e nome sarebbe troppo lunga, pertanto si concorda con l'utente di usare il nominativo '**dicembrinosempr**':

dicembrinosempr OK



E essendo già stata prevista la classificazione '**DOCENTI0**' la si seleziona e si procede. Si passa così alla richiesta della descrizione completa del docente:



prof. Dicembrino Sempronio economia aziendale 2012/2013 [*Invio*]

```
Adding user dicembrinosempr...
Adding new group dicembrinosempr (1004).
Adding new user dicembrinosempr (1004) with group
 dicembrinosempr.
Creating home directory /home/DOCENTI0/dicembrinosempr.
Copying files from /etc/skel
```

```
.----------------------New password-------------------------.
| [en] Please insert the new password for user              |
| "dicembrinosempr" with home directory                     |
| "/home/DOCENTI0/dicembrinosempr". Please insert at least  |
| 8 characters.                                             |
|                                                           |
| [it] Inserire la nuova parola d'ordine per l'utente       |
| "dicembrinosempr" che dispone della directory personale   |
| "/home/DOCENTI0/dicembrinosempr". Si prega di inserire    |
| almeno 8 caratteri.                                       |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'

.----------------------New password-------------------------.
| [en] Please insert again the new password for user        |
| "dicembrinosempr" with home directory                     |
| "/home/DOCENTI0/dicembrinosempr".                         |
|                                                           |
| [it] Inserire nuovamente la parola d'ordine per l'utente  |
| "dicembrinosempr" che dispone della directory personale   |
| "/home/DOCENTI0/dicembrinosempr".                         |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

Al termine, dopo la conferma dell'inserimento della parola d'ordine, non dovendo inserire altri utenti, basta concludere selezionando il pulsante grafico CANCEL.

Quando si tenta di inserire un nominativo utente molto lungo, è probabile che l'operazione si concluda ugualmente con successo, ma ciò avviene perché il programma riduce automaticamente il nome ai primi 15 caratteri. Pertanto, se si inserisce il nominativo «dicembrinosempronio» si ottiene in pratica l'utente «dicembrinosempr». Di questo occorre tenerne conto, perché poi gli utenti chiedono aiuto quando non riescono ad accedere al sistema; in tal caso basta dire loro di riprovare con i soli primi 15 caratteri del nominativo presunto.

Controllo dei servizi

Presso l'elaboratore 172.17.1.254 sono presenti diversi servizi per la rete locale, compreso quello di stampa e un proxy HTTP trasparente. L'accesso a questi servizi può essere controllato, per impedire ad alcuni elaboratori di stampare o di accedere a servizi HTTP esterni con il navigatore. Ciò che si imposta in questo modo, rimane, anche all'eventuale riavvio dell'elaboratore 172.17.1.254, pertanto è necessario sapere come ripristinare o comunque regolare tali servizi.

Dal menù si seleziona la voce *printer access* per il controllo dell'utilizzo della stampante, oppure la voce *proxy access* per il controllo dell'accesso ai servizi esterni HTTP. In entrambi i casi si ottiene un elenco degli elaboratori che possono essere abilitati o disabilitati; l'esempio seguente riguarda il caso del proxy HTTP:


```
.----------HTTP proxy access permissions-------------.
| Please, select or deselect who can access to the   |
| HTTP proxy:                                        |
| .------------------------------------------------. |
| | [ ] DENY_ALL       reset to no access allowed  | |
| | [ ] ALLOW_ALL      reset to all access allowed | |
| | [X] 172.17.1.1     allow_172.21.1.1            | |
| | [ ] 172.17.1.10    allow_172.21.1.10           | |
| | [X] 172.17.1.11    allow_172.21.1.11           | |
| | [ ] 172.17.1.12    allow_172.21.1.12           | |
| | [ ] 172.17.1.13    allow_172.21.1.13           | |
| | [ ] 172.17.1.14    allow_172.21.1.14           | |
| | [ ] 172.17.1.15    allow_172.21.1.15           | |
| | [ ] 172.17.1.16    allow_172.21.1.16           | |
| | [ ] 172.17.1.17    allow_172.21.1.17           | |
| | [ ] 172.17.1.18    allow_172.21.1.18           | |
| | [ ] 172.17.1.19    allow_172.21.1.19           | |
| | [X] 172.17.1.2     allow_172.21.1.2            | |
| | [ ] 172.17.1.20    allow_172.21.1.20           | |
| | [ ] 172.17.1.21    allow_172.21.1.21           | |
| '-----v(+)---------------------------------------' |
|----------------------------------------------------|
|            <  OK  >         <Cancel>              |
'----------------------------------------------------'
```

Per selezionare o deselezionare una voce, basta premere la barra spaziatrice quando quella che si desidera è evidenziata; per confermare le selezioni fatte, si seleziona il pulsante OK. Nell'esempio è in evidenza la richiesta di attivare il collegamento per gli elaboratori 172.17.1.1, 172.17.1.2 e 172.17.1.11.

Controllo del numero massimo di pagine stampabili per volta

Presso ogni singolo elaboratore è configurato un numero massimo di pagine stampabili per volta: in condizioni normali è possibile produrre un massimo di 11 pagine per stampa. Nel caso fosse necessario stampare una quantità maggiore di pagine, vanno divise le richieste di stampa in blocchi di quantità inferiore o uguale a 11.

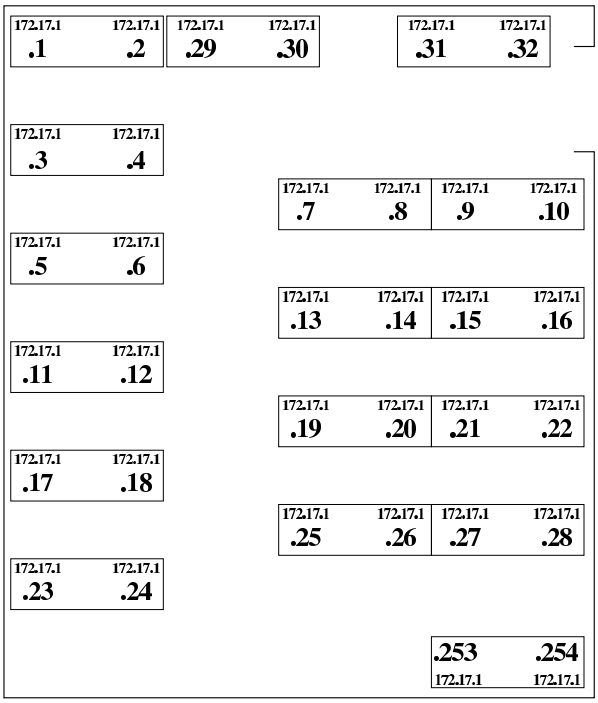
Utilizzo di elaboratori estranei al laboratorio

Se per qualche ragione devono essere usati nel laboratorio degli elaboratori diversi da quelli previsti, collegandoli alla rete, è necessario provvedere alla configurazione dell'interfaccia di rete e all'instradamento necessari. In condizioni normali va utilizzato il protocollo DHCP per la configurazione automatica; tuttavia, nel caso si debba procedere in modo manuale, oltre a chiedere al responsabile l'uso di un indirizzo IPv4, occorre sapere che l'indirizzo della rete locale è 172.17.0.0, con maschera di rete 255.255.0.0 e che il router per accedere alla rete esterna dall'interno del laboratorio è 172.17.1.254.

Avvio e arresto degli elaboratori del laboratorio 199
Utenze 200
Registri 200
Console e utilizzo di applicazioni grafiche 200
Stampa 201
Accesso a Internet 201
Licenze 202
The Debian Free Software Guidelines (DFSG) 202

Il laboratorio è distribuito secondo la piantina che si può vedere nella figura successiva. Ogni elaboratore è distinto secondo il suo indirizzo IPv4 e possono essere usati dagli utenti tutti quelli che hanno indirizzi da 172.17.1.1 a 172.17.1.32, perché i due elaboratori 172.17.1.254 e 172.17.1.253 mettono a disposizione dei servizi indispensabili per la rete e non possono sostenere carichi ulteriori. Il sistema operativo utilizzato negli elaboratori è GNU/Linux. Gli elaboratori con indirizzi 172.17.1.254 e 172.17.1.253 hanno anche un indirizzo alternativo, corrispondente, rispettivamente, a 192.168.0.71 e 192.168.0.81.

Figura u42.1. Piantina del laboratorio.



Avvio e arresto degli elaboratori del laboratorio

Nel laboratorio rimane sempre acceso l'elaboratore 172.17.1.254, perché offre servizi importanti per tutta la rete didattica e perché nelle ore di non utilizzo può eseguire le operazioni di manutenzione giornaliera di registri, indici e di scansione antivirus nei dati personali degli utenti.

Gli elaboratori destinati agli utenti, vanno accesi e spenti all'occorrenza, ma in ogni caso è previsto un piano di spegnimento automatico giornaliero, per maggiore sicurezza. Gli elaboratori destinati agli utenti utilizzano un sistema operativo avviato dalla rete (fornito dal servente 172.17.1.254), così da non dipendere dal disco fisso locale; pertanto, possono essere spenti anche senza accortezze, in caso di necessità.

Se l'elaboratore 172.17.1.254 non è in funzione regolarmente, per qualunque ragione, gli altri elaboratori non sono utilizzabili.

Utenze

Per poter utilizzare qualunque postazione del laboratorio, è necessario disporre di un'utenza personale, a cui è associata una parola d'ordine di riconoscimento:

`login:` **`rossimario`** [*Invio*]

`Password:` **`segretissimo`** [*Invio*]

Per sicurezza, l'inserimento della parola d'ordine viene fatto all'oscuro, senza nemmeno poter vedere quanti tasti sono stati premuti.

Le utenze devono essere personali e possono essere create solo da un «amministratore», ovvero da una persona che abbia i privilegi necessari per questo compito. La parola d'ordine viene decisa dallo stesso utente che deve usarla successivamente per identificarsi, il quale deve anche avere la cura di ricordarsela e di mantenerla segreta.

In base all'organizzazione del laboratorio, per poter cambiare la propria parola d'ordine è necessario chiedere aiuto all'amministratore.

Gli utenti del laboratorio devono essere ben consapevoli del fatto che alcune delle attività svolte sono annotate in un registro elettronico, disponibile pubblicamente nell'ambito della rete locale. I file di tale registro vengono conservati (salvo malfunzionamenti), per un tempo discreto e potrebbero servire per verificare l'utilizzo corretto del laboratorio stesso da parte di chi vi accede, benché tale ricerca sia comunque di una certa complessità. Anche per questa ragione è molto importante mantenere segreta la propria parola d'ordine di identificazione, inoltre non conviene abbandonare, anche solo temporaneamente, un elaboratore (o un terminale) lasciando attiva la propria sessione di lavoro.

Ogni utente dispone di una directory personale, ovvero di uno spazio per i propri dati personali. In base alle caratteristiche tecniche degli elaboratori disponibili, lo spazio concesso a ogni utente è di 50 000 000 byte. L'utilizzo dello spazio da parte di ogni utente deve essere tenuto sotto controllo, perché i programmi più comuni (come quelli di navigazione o quelli di automazione dell'ufficio) creano spesso in modo automatico dei file nella directory personale, in modo da conservare la configurazione particolare dell'utente. Gli utenti che si accorgono del problema e, per qualche ragione, non sono in grado di ridurre lo spazio utilizzato, devono chiedere aiuto.

In generale, pur non essendoci alcuna intenzione di danneggiare gli utenti del laboratorio, non è possibile dare alcuna garanzia che i dati vengano conservati integri nel tempo; inoltre, all'inizio di ogni anno scolastico vengono azzerate tutte le utenze degli studenti, o per lo meno quelle classificate come tali. Pertanto, gli utenti che hanno la necessità di conservare i propri dati attraverso gli anni scolastici devono organizzarsi attraverso delle copie di sicurezza.

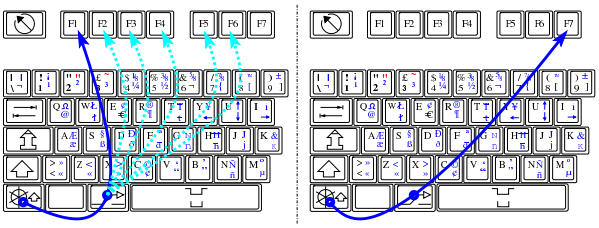
Registri

Per motivi didattici, dall'interno del laboratorio è possibile accedere all'indirizzo *http://172.17.1.254/cgi-bin/var_log*, dal quale è possibile leggere la maggior parte dei file delle registrazioni raccolti dall'elaboratore 172.17.1.254.

Console e utilizzo di applicazioni grafiche

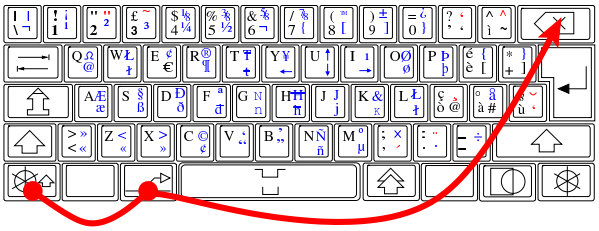
In generale le postazioni di lavoro normali attivano automaticamente il sistema grafico dopo l'identificazione dell'utente. Tuttavia, sono disponibili anche le console virtuali tradizionali (a caratteri, senza grafica). Le console virtuali sono raggiungibili con le combinazioni di tasti [*Ctrl Alt F1*], [*Ctrl Alt F2*], fino a [*Ctrl Alt F6*]; inoltre, la combinazione [*Ctrl Alt F7*] riporta alla sessione di lavoro grafica.

Figura u42.2. Selezione delle sessioni di lavoro: a sinistra le combinazioni di tasti per le console virtuali; a destra la combinazione per la sessione grafica.



In caso di necessità, durante il funzionamento in modalità grafica, è possibile eliminare il processo elaborativo del sistema grafico X attraverso la combinazione di tasti [*Ctrl Alt Backspace*] (ovvero [*Ctrl Alt* <- - -]).

Figura u42.3. Uso della combinazione di tasti [*Ctrl Alt Backspace*].



Il sistema grafico è organizzato in modo da non avere «oggetti» (*gadget*) superflui. Tra le altre cose, ciò consente di ottenere il massimo delle prestazioni dall'elaboratore senza sprechi. Per questa ragione, il puntatore grafico del mouse non mostra clessidre o altro per indicare l'aumento dell'attività del sistema operativo; quindi, quando si avvia un'applicazione, occorre aspettare un momento prima di poterla vedere apparire.

Stampa

Sono disponibili due stampanti, collegate rispettivamente all'elaboratore 172.17.1.254 e 172.17.1.253. La stampa presso tutti gli elaboratori avviene in modo predefinito utilizzando la stampante collegata all'elaboratore 172.17.1.254; mentre la stampante ausiliaria viene utilizzata solo durante l'uso dell'elaboratore 172.17.1.253. In caso di avaria della stampante principale, la stampante ausiliaria viene collegata all'elaboratore 172.17.1.254.

Accesso a Internet

L'accesso a Internet avviene attraverso l'elaboratore con indirizzo 172.17.1.254, il quale svolge il compito di router e di proxy trasparente. La funzionalità di proxy include anche un filtro sommario di siti e di URI.

La censura di una pagina si manifesta attraverso una schermata simile a quella della figura successiva.

Figura u42.4. Messaggio di avvertimento della censura di un indirizzo o di una pagina, generato dal servizio proxy.

Here is result of your request:

Denied by ACL: BAN_by_word_sex

Probably the requested site doesn't answer or it timed out. Check the address or try again.

Administrator

Generated by oops at nanohost:3128

Licenze

«

Salvo indicazione diversa, nel laboratorio viene usato software libero che risponde alle linee guida DFSG (*Debian free software guidelines*), citate nella sezione i42.7.1 (il testo originale si trova presso *http://www.debian.org/social_contract*); oltre al software è disponibile anche documentazione che, pur non essendo modificabile, può essere riprodotta senza oneri.

The Debian Free Software Guidelines (DFSG)[1]

«

1. **Free Redistribution**

   The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

2. **Source Code**

   The program must include source code, and must allow distribution in source code as well as compiled form.

3. **Derived Works**

   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. **Integrity of The Author's Source Code**

   The license may restrict source-code from being distributed in modified form **_only_** if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (*This is a compromise. The Debian group encourages all authors not to restrict any files, source or binary, from being modified.*)

5. **No Discrimination Against Persons or Groups**

   The license must not discriminate against any person or group of persons.

6. **No Discrimination Against Fields of Endeavor**

   The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. **Distribution of License**

   The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. **License Must Not Be Specific to Debian**

   The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

9. **License Must Not Contaminate Other Software**

   The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

10. **Example Licenses**

    The "**GPL**", "**BSD**", and "**Artistic**" licenses are examples of licenses that we consider "*free*".

Figura u42.5. Piantina del laboratorio per le annotazioni riferite alle varie postazioni.

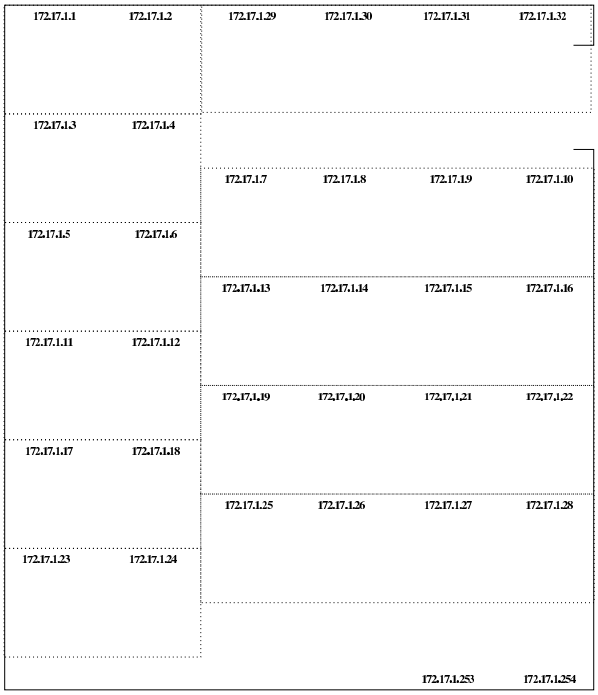


Figura u42.6. Scheda per le annotazioni riferite all'uso del laboratorio.

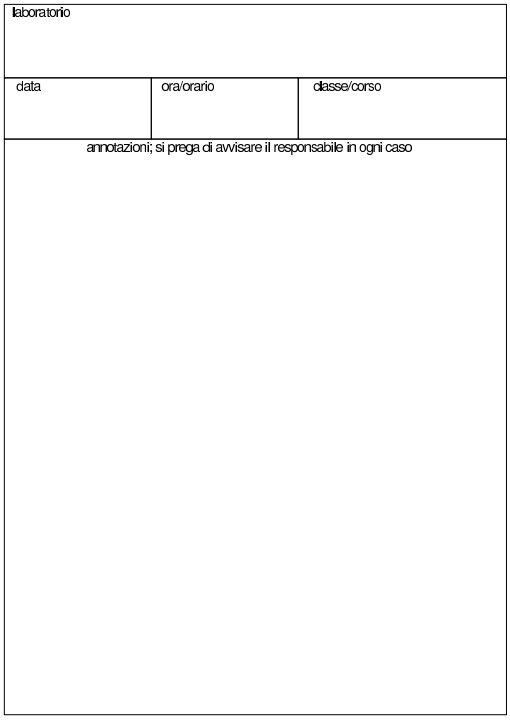
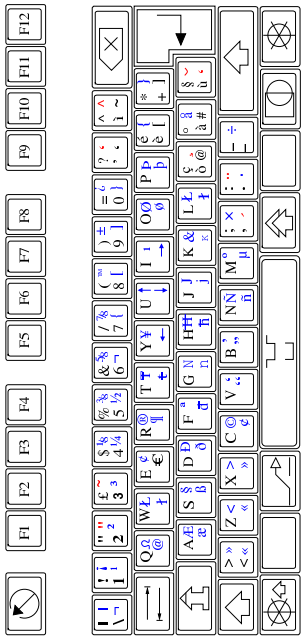


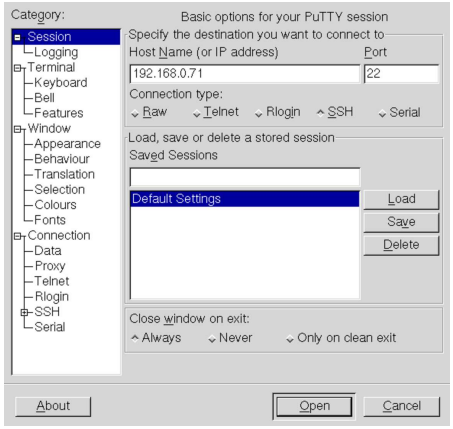
Figura u42.7. Mappa per l’uso della tastiera. Si può stampare una mappa di questa tastiera a partire da *allegati/mappa-della-tastiera-italiana.ps* .



[1] Citazione da *http://www.debian.org/social_contract*

Procedura per l’aggiunta di un utente 206
Procedura per la modifica di una parola d’ordine 210
Controllo dell’accesso all’esterno 212
Configurazione di MS-Windows XP Professional per utilizzare le utenze centralizzate 212
Configurazione di MS-Windows 7 per utilizzare le utenze centralizzate 213
Configurazione per poter utilizzare anche un sistema GNU/Linux 217
Eliminazione delle utenze 218

Gli insegnanti che hanno ottenuto la facoltà di intervenire in questo modo, hanno un’utenza amministrativa (corrispondente solitamente al solo cognome), con la quale possono connettersi all’elaboratore 172.17.1.254, ovvero 192.168.0.71, attraverso il programma PuTTY, usando preferibilmente il protocollo SSH (cifrato), come si vede nella figura:



L’amministratore deve quindi introdurre il proprio nominativo-utente e la parola d’ordine, quindi ottiene il menù di funzioni che gli sono state concesse:

```
.--------------------Admin menu-----------------------.
| Remote admin limited menu                           |
| .-------------------------------------------------. |
| |   adduser       Add a new user                  | |
| |   passwd        Change a user's password        | |
| |   description   Change a user's description     | |
| |   user info     Show user info                  | |
| |   home info     Show home directory info        | |
| |   quota report  Show sorted user's quota        | |
| |   nis-make      Rebuild NIS database            | |
| |   add machine   Add a new MS-Windows computer   | |
| |   proxy access  HTTP proxy access permissions   | |
| |   admin passwd  Change your password            | |
| |   custom1       Custom script 1                 | |
| |   custom2       Custom script 2                 | |
| |   custom3       Custom script 3                 | |
| |   exit          Quit                            | |
| '-------------------------------------------------' |
|                                                     |
|                                                     |
|-----------------------------------------------------|
|            <  OK  >       <Cancel>                 |
'-----------------------------------------------------'
```

Attraverso questo menù è possibile, in particolare, aggiungere un’utenza, cambiare la parola d’ordine di un utente che ne fa richiesta e verificare la configurazione delle utenze.

Tabella u43.3. Descrizione delle funzioni disponibili agli amministratori.

Funzione	Descrizione
`adduser`	Consente di aggiungere un'utenza al sistema.
`passwd`	Consente di modificare una parola d'ordine.
`description`	Consente di modificare la descrizione estesa di un'utenza.
`user info` `home info`	Consente di avere informazioni su un'utenza.
`quota report`	Mostra l'elenco delle quote degli utenti, ordinata in modo decrescente per quantità di spazio utilizzato.
`nis-make`	Ricostruisce la base di dati NIS a partire dalle informazioni aggiornate delle utenze.
`add machine`	Aggiunge un'utenza speciale, corrispondente a un elaboratore MS-Windows, per consentirgli l'accesso con Samba.
`proxy access`	Permette di controllare l'accesso alla rete esterna tramite il protocollo HTTP, con la configurazione del servizio proxy.
`admin passwd`	Consente all'amministratore di cambiare la propria parola d'ordine necessaria per accedere.
`custom1`	Ricostruisce i collegamenti tra studenti e insegnanti, per lo scambio di verifiche e di strumenti didattici.
`custom2` `custom3`	Consente di controllare l'accesso alle unità di memorizzazione esterne.
`custom3` `custom3`	Consente di controllare l'accesso a risorse esterne, tramite il proxy HTTP, nel proprio laboratorio o in altri, se configurati in modo appropriato.
`exit`	Esce dal menù.

Procedura per l'aggiunta di un utente

«

Dal momento che si prevede la presenza simultanea di un gran numero di utenze, si richiede a chi interviene per aggiungerne di nuove di farlo con un certo ordine.

Per prima cosa **si conviene che il nominativo scelto dall'utente cominci con il cognome** e continui, possibilmente, con il nome (per esempio «rossimario»), tenendo conto che si possono usare al massimo 15 caratteri alfabetici e numerici (lettere dalla «a» alla «z», minuscole, cifre da zero a nove).

Una volta inserito il nominativo, è necessario stabilire una sigla che viene usata per classificare l'utenza. Nel caso degli studenti che usano comunemente il laboratorio, va usata la sigla della classe (compresa la sezione e una lettera per distinguere il corso di studi) seguita dall'anno scolastico. La tabella seguente riepiloga alcuni esempi.

Tabella u43.4. Esempi di sigle da usare per raggruppare le utenze secondo il contesto per il quale vengono create.

Esempio	Contesto
`5A1213`	Utenze annuali, riferite agli alunni di 5A, create nell'anno scolastico 2012/2013, dove non è necessario specificare il corso di studi.
`4AS1213`	Utenze annuali, riferite agli alunni di 4A, corso «S», create nell'anno scolastico 2012/2013.
`DOCENTI0`	Utenze riferite ai docenti, da conservare attraverso gli anni scolastici.
`ITP00000`	Utenze riferite agli insegnanti tecnico-pratici, da conservare attraverso gli anni scolastici.
`TECNICI0`	Utenze riferita ai tecnici, da conservare attraverso gli anni scolastici.
`COLLSCOL`	Utenze riferite ai tecnici, da conservare attraverso gli anni scolastici.
`SEGRETER`	Utenze riferite al personale di segreteria, da conservare attraverso gli anni scolastici.

Esempio	Contesto
`DSGA0000`	Utenza riferita al direttore dei servizi generali e amministrativi, da conservare attraverso gli anni scolastici.
`DIRIGENT`	Utenza riferita al dirigente scolastico, da conservare attraverso gli anni scolastici.
`ESTERNI0`	Utenti che non fanno parte dell'istituto, ma che sono ammessi ad accedere alla rete didattica per qualche motivo.

Quando viene richiesto di inserire un utente, il programma che si occupa di questo consente di aggiungere dei dati ulteriori all'interno di un campo aggiuntivo. Ciò va usato per indicare i dati significativi dell'utenza, in base al contesto di utilizzo. Seguono due esempi; il primo riferito all'utente Mario Rossi che è uno studente di 5A «igea», il secondo riferito al professor Sempronio Dicembrino che è docente di economia aziendale:

```
.----------------------Add a new user-----------------------.
| [en] Please insert the new user name (only lower case    |
| letters and numbers, minimum 8 and max 15 characters,    |
| the first character must be a letter).                   |
|                                                          |
| [it] Inserire il nominativo utente, composto             |
| preferibilmente da cognome e nome attaccati, ed          |
| eventualmente l'anno di nascita, per esempio             |
| "rossimario1990" (si possono usare solo lettere          |
| minuscole e cifre numeriche per un minimo di 8 e fino a  |
| un massimo di 15 caratteri, ma il primo carattere deve   |
| essere una lettera).                                     |
| |........9012345|                                        |
| .------------------------------------------------------. |
| |                                                      | |
| '------------------------------------------------------' |
|                                                          |
|----------------------------------------------------------|
|                 <  OK  >            <Cancel>             |
'----------------------------------------------------------'
```

`rossimario` OK

```
.--------------------Classify user-----------------------.
| [en] Please select a hierarchy name for the user: it   |
| will be used as an intermediate directory after        |
| "/home/".                                              |
|                                                        |
| [it] Inserire la classificazione dell'utente: viene    |
| usata come directory intermedia dopo "/home/".         |
| .--------------------------------------------------. |
| |              1AI1213   /home/1AI1213             | |
| |              1AM1213   /home/1AM1213             | |
| |              1AS1213   /home/1AS1213             | |
| |              1BI1213   /home/1BI1213             | |
| |              1BM1213   /home/1BM1213             | |
| |              1BS1213   /home/1BS1213             | |
| |              1CI1213   /home/1CI1213             | |
| |              1CM1213   /home/1CM1213             | |
| |              1DI1213   /home/1DI1213             | |
| |              DOCENTI0  /home/DOCENTI0/           | |
| |              --------  ----------------------    | |
| |              new       add a new one             | |
| '--------------v(+)-----------------------------------' |
|                                                        |
|                                                        |
|--------------------------------------------------------|
|               <  OK  >        <Cancel>                |
'--------------------------------------------------------'
```

Non essendo già stata prevista la classificazione '**`5AI1213`**', si seleziona la voce '**`new`**' per passare all'inserimento manuale:

```
.----------------------Classify user-----------------------.
| [en] Please insert a hierarchy name for the user: it     |
| will be used as an intermediate directory after "/home/".|
| Please insert only upper case letters and digits (A-Z,   |
| 0-9; min 4, max 8 characters).                           |
|                                                          |
| [it] Inserire la classificazione dell'utente: viene      |
| usata come directory intermedia dopo "/home/".           |
| Inserire solo lettere maiuscole e cifre numeriche (A-Z,  |
| 0-9; min 4, max 8 caratteri).                            |
| |....5678|                                               |
| .------------------------------------------------------. |
| |2012                                                  | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|                 <  OK  >           <Cancel>              |
'----------------------------------------------------------'
```

Viene cancellato il valore predefinito e viene assegnata la sigla '**5AI1213**':

[*Canc*][*Canc*][*Canc*][*Canc*]

5AI1213 OK

```
.--------------------Full user name----------------------.
| [en] Please insert the user full name and maybe some    |
| more data to identify it.                               |
|                                                         |
| [it] Inserire la descrizione dettagliata dell'utente,   |
| per poterlo identificare con precisione.                |
| .-------------------------------------------------. |
| |                                                 | |
| '-------------------------------------------------' |
|---------------------------------------------------------|
|               <  OK  >        <Cancel>                  |
'---------------------------------------------------------'
```

Vengono richiesti alcuni dati addizionali, che è bene compilare per poter individuare correttamente l'utente:

studente Rossi Mario 5A igea 2012/2013 [*Invio*]

```
Adding user 'rossimario'...
Adding new group 'rossimario' (1000).
Adding new user 'rossimario' (1000) with group 'rossimario'.
Creating home directory '/home/5AI1213/rossimario'.
Copying files from '/etc/skel'
```

Al termine viene richiesto di inserire per due volte la parola d'ordine, cosa che deve essere fatta direttamente dalla persona per la quale si crea l'utenza.

```
.----------------------New password------------------------.
| [en] Please insert the new password for user "rossimario"|
| with home directory                                      |
| "/home/5A1213/rossimario". Please insert at least 8      |
| characters.                                              |
|                                                          |
| [it] Inserire la nuova parola d'ordine per l'utente      |
| "rossimario" che dispone della directory personale       |
| "/home/5A1213/rossimario".                               |
| Si prega di inserire almeno 8 caratteri.                 |
|  <-MIN-->                                                |
| .------------------------------------------------------. |
| |***********                                         | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|                <  OK  >          <Cancel>                |
'----------------------------------------------------------'
```

L'inserimento corrisponde alla visualizzazione di una serie di asterischi.


```
.----------------------New password------------------------.
| [en] Please insert again the new password for user       |
| "rossimario" with home directory "/home/5A1213/rossimario".|
|                                                          |
| [it] Inserire nuovamente la parola d'ordine per l'utente |
| "rossimario" che dispone della directory personale       |
| "/home/5A1213/rossimario".                               |
|  <-MIN->                                                 |
| .------------------------------------------------------. |
| |***********                                         | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|                <  OK  >          <Cancel>                |
'----------------------------------------------------------'
```

Dopo l'inserimento, per due volte, della parola d'ordine, se tutto è stato fatto senza errori, soprattutto se la parola d'ordine è stata inserita correttamente, l'operazione è completa e si può procedere con l'utente successivo:

```
.----------------------Add a new user----------------------.
| [en] Please insert the new user name (only lower case    |
| letters and numbers, minimum 8 and max 15 characters,    |
| the first character must be a letter).                   |
|                                                          |
| [it] Inserire il nominativo utente, composto             |
| preferibilmente da cognome e nome attaccati, ed          |
| eventualmente l'anno di nascita, per esempio             |
| "rossimario1990" (si possono usare solo lettere          |
| minuscole e cifre numeriche per un minimo di 8 e fino a  |
| un massimo di 15 caratteri, ma il primo carattere deve   |
| essere una lettera).                                     |
| |........9012345|                                      |
| .------------------------------------------------------. |
| |                                                      | |
| '------------------------------------------------------' |
|----------------------------------------------------------|
|               <  OK  >           <Cancel>               |
'----------------------------------------------------------'
```

Passando all'inserimento dell'utente «Sempronio Dicembrino», la somma di cognome e nome sarebbe troppo lunga, pertanto si concorda con l'utente di usare il nominativo '**dicembrinosempr**':

dicembrinosempr OK

```
.----------------------Classify user-----------------------.
| [en] Please select a hierarchy name for the user: it     |
| will be used as an intermediate directory after          |
| "/home/".                                                |
|                                                          |
| [it] Inserire la classificazione dell'utente: viene      |
| usata come directory intermedia dopo "/home/".           |
| .------------------------------------------------------. |
| |              1AI1213   /home/1AI1213                | |
| |              1AM1213   /home/1AM1213                | |
| |              1AS1213   /home/1AS1213                | |
| |              1BI1213   /home/1BI1213                | |
| |              1BM1213   /home/1BM1213                | |
| |              1BS1213   /home/1BS1213                | |
| |              1CI1213   /home/1CI1213                | |
| |              1CM1213   /home/1CM1213                | |
| |              1DI1213   /home/1DI1213                | |
| |              DOCENTI0  /home/DOCENTI0/              | |
| |              --------  ----------------------      | |
| |              new       add a new one               | |
| '--------------v(+)-----------------------------------' |
|                                                          |
|                                                          |
|----------------------------------------------------------|
|               <  OK  >        <Cancel>                  |
'----------------------------------------------------------'
```

E essendo già stata prevista la classificazione '**DOCENTI0**' la si seleziona e si procede. Si passa così alla richiesta della descrizione completa del docente:

```
.--------------------Full user name----------------------.
| [en] Please insert the user full name and maybe some    |
| more data to identify it.                               |
|                                                         |
| [it] Inserire la descrizione dettagliata dell'utente,   |
| per poterlo identificare con precisione.                |
| .-----------------------------------------------------. |
| |                                                     | |
| '-----------------------------------------------------' |
|---------------------------------------------------------|
|               <  OK  >        <Cancel>                  |
'---------------------------------------------------------'
```

prof. Dicembrino Sempronio economia aziendale 2012/2013 [*Invio*]

```
Adding user dicembrinosempr...
Adding new group dicembrinosempr (1004).
Adding new user dicembrinosempr (1004) with group
 dicembrinosempr.
Creating home directory /home/DOCENTI0/dicembrinosempr.
Copying files from /etc/skel
```

```
.----------------------New password-------------------------.
| [en] Please insert the new password for user              |
| "dicembrinosempr" with home directory                     |
| "/home/DOCENTI0/dicembrinosempr". Please insert at least  |
| 8 characters.                                             |
|                                                           |
| [it] Inserire la nuova parola d'ordine per l'utente       |
| "dicembrinosempr" che dispone della directory personale   |
| "/home/DOCENTI0/dicembrinosempr". Si prega di inserire    |
| almeno 8 caratteri.                                       |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

```
.----------------------New password-------------------------.
| [en] Please insert again the new password for user        |
| "dicembrinosempr" with home directory                     |
| "/home/DOCENTI0/dicembrinosempr".                         |
|                                                           |
| [it] Inserire nuovamente la parola d'ordine per l'utente  |
| "dicembrinosempr" che dispone della directory personale   |
| "/home/DOCENTI0/dicembrinosempr".                         |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

Al termine, dopo la conferma dell'inserimento della parola d'ordine, non dovendo inserire altri utenti, basta concludere selezionando il pulsante grafico CANCEL.

Quando si tenta di inserire un nominativo utente molto lungo, è probabile che l'operazione si concluda ugualmente con successo, ma ciò avviene perché il programma riduce automaticamente il nome ai primi 15 caratteri. Pertanto, se si inserisce il nominativo «dicembrinosempronio» si ottiene in pratica l'utente «dicembrinosempr». Di questo occorre tenerne conto, perché poi gli utenti chiedono aiuto quando non riescono ad accedere al sistema; in tal caso basta dire loro di riprovare con i soli primi 15 caratteri del nominativo presunto.

Procedura per la modifica di una parola d'ordine

Gli studenti sono spesso smemorati ed è facile che dimentichino la parola d'ordine necessaria per accedere ai propri dati. Per questo, gli insegnanti che hanno facoltà amministrative, devono poter consentire agli studenti di modificare la loro parola d'ordine, facendo però attenzione che si tratti effettivamente dell'utente corretto.

Mentre l'aggiunta delle utenze può anche essere affidata agli studenti, in modo che ognuno aggiunga la propria, la modifica della parola d'ordine deve essere sempre guidata dall'amministratore, il quale deve controllare che si tratti effettivamente di un nominativo associato alla classe a cui dovrebbe appartenere. Infatti, potrebbe succedere che lo studente dimentichi il proprio nominativo utente corretto (e non la parola d'ordine), quando nella scuola ci possono essere delle omonimie che hanno richiesto di usare delle piccole varianti nei nominativi rispetto alla regola generale del cognome+nome.

La prima maschera che viene proposta all'amministratore, richiede l'inserimento del nominativo-utente, per il quale va cambiata la parola d'ordine:

```
.---------------Change password------------------.
| [en] Please insert the user name who have to   |
| change the password.                           |
|                                                |
| [it] Inserire il nominativo utente per il      |
| quale si deve cambiare la parola d'ordine.     |
| .--------------------------------------------. |
| |                                            | |
| '--------------------------------------------' |
|------------------------------------------------|
|          <  OK  >      <Cancel>                |
'------------------------------------------------'
```

rossimario OK

A questo punto viene chiesto di digitare subito la nuova parola d'ordine, ma, come si vede dall'esempio, si può verificare il percorso della directory personale dell'utente relativo, dove è indicata la classe a cui questo appartiene. Ciò dovrebbe consentire di fare confusione tra nominativi-utente di studenti omonimi, appartenenti a classi differenti.

```
.----------------------New password-------------------------.
| [en] Please insert the new password for user "rossimario"|
| with home directory                                       |
| "/home/5A1213/rossimario". Please insert at least 8       |
| characters.                                               |
|                                                           |
| [it] Inserire la nuova parola d'ordine per l'utente       |
| "rossimario" che dispone della directory personale        |
| "/home/5A1213/rossimario". Si prega di inserire almeno 8  |
| caratteri.                                                |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

Logicamente, la digitazione della parola d'ordine è compito dell'utente che deve cambiarla.

digitazione_all'oscuro OK

```
.----------------------New password-------------------------.
| [en] Please insert again the new password for user        |
| "rossimario" with home directory "/home/5A1213/rossimario".|
|                                                           |
| [it] Inserire nuovamente la parola d'ordine per l'utente  |
| "rossimario" che dispone della directory personale        |
| "/home/5A1213/rossimario".                                |
|  <-MIN-->                                                 |
| .-------------------------------------------------------. |
| |************                                           | |
| '-------------------------------------------------------' |
|-----------------------------------------------------------|
|                <  OK  >          <Cancel>                 |
'-----------------------------------------------------------'
```

digitazione_all'oscuro OK

Controllo dell'accesso all'esterno

Per poter controllare l'accesso da parte degli studenti alla rete esterna (Internet), è necessario che venga utilizzato come *gateway* l'indirizzo 172.17.1.254.

La configurazione del servizio proxy è tale da escludere l'accesso a siti che contengono nel loro indirizzo URI alcune parole chiave ritenute imbarazzanti; inoltre, dovrebbe impedire lo scarico di file audio-visuali.

Attraverso la funzione '**`proxy access`**' del menù amministrativo, è possibile controllare l'accessibilità all'esterno da parte delle postazioni. In altri termini, con la funzione '**`proxy access`**' è possibile abilitare o disabilitare l'accesso a Internet dai laboratori, per le postazioni configurate in modo da avvalersi del router 172.17.1.254.

Figura u43.21. Le voci che hanno una «X» rappresentano gli elaboratori che possono accedere alla rete esterna.

```
.------------------proxy access--------------------.
| Please, select or deselect who can access to the |
| HTTP proxy:                                      |
| .----------------------------------------------. |
| | [ ] DENY_ALL     reset to no access allowed  | |
| | [ ] ALLOW_ALL    reset to all access allowed | |
| | [*] 172.17.4.26  allow_172.17.4.26           | |
| | [*] 172.17.4.27  allow_172.17.4.27           | |
| | [*] 172.17.4.28  allow_172.17.4.28           | |
| | [*] 172.17.4.30  allow_172.17.4.30           | |
| | [ ] 172.17.4.1   allow_172.17.4.1            | |
| | [ ] 172.17.4.2   allow_172.17.4.2            | |
| | [ ] 172.17.4.3   allow_172.17.4.3            | |
| | [ ] 172.17.4.4   allow_172.17.4.4            | |
| | [ ] 172.17.4.5   allow_172.17.4.5            | |
| | [ ] 172.17.4.6   allow_172.17.4.6            | |
| | [ ] 172.17.4.7   allow_172.17.4.7            | |
| '------.(+)-----------------------------------' |
|                                                  |
|                                                  |
|                                                  |
|--------------------------------------------------|
|             <  OK  >       <Cancel>             |
'--------------------------------------------------'
```

Configurazione di MS-Windows XP Professional per utilizzare le utenze centralizzate

Per configurare MS-Windows XP in modo da poter utilizzare le utenze centralizzate, è necessario associare l'elaboratore al dominio «INF». Per questo occorre abilitare l'accesso dell'elaboratore presso il servente, pertanto, dal menù di amministrazione a cui si accede con l'ausilio di PuTTY (come già descritto per la gestione delle utenze), si utilizza la voce *add machine*:

```
.--Add a new Win machine-----.
| Please insert the new Win  |
| machine name:              |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK   >   <Cancel>     |
'----------------------------'
```

A titolo di esempio si considera che si tratti dell'elaboratore con nome «PC07x». Il nome va inserito usando solo lettere minuscole:

`pc07x` OK

```
.Full Win machine description--.
| Please insert the machine    |
| full description.            |
| .--------------------------. |
| |                          | |
| '--------------------------' |
|------------------------------|
|   <  OK   >   <Cancel>       |
'------------------------------'
```

È il caso di indicare il nome del laboratorio o della stanza in cui si trova:

`Laboratorio informatica` ... OK

Una volta aggiunto l'elaboratore all'elenco di quelli ammessi al servizio, si torna alla configurazione di MS-Windows XP, per l'associazione del dominio «INF»; per questo occorre agire con i privilegi dell'utente '**`Administrator`**'.

- *Pannello di controllo*
  - *Prestazioni e manutenzione*
  - *Sistema*
    - *Nome computer*

A questo punto compare una maschera simile a quella della figura successiva, dove occorre selezionare il pulsante grafico CAMBIA.

Figura u43.24. Proprietà del sistema: cambiamento del nome o dell'associazione a un dominio.

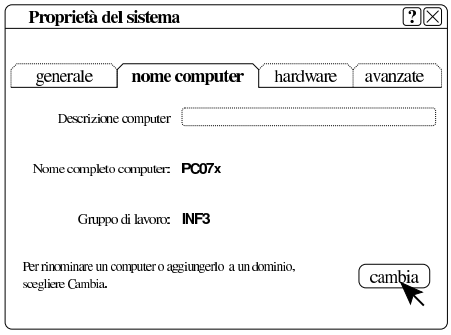
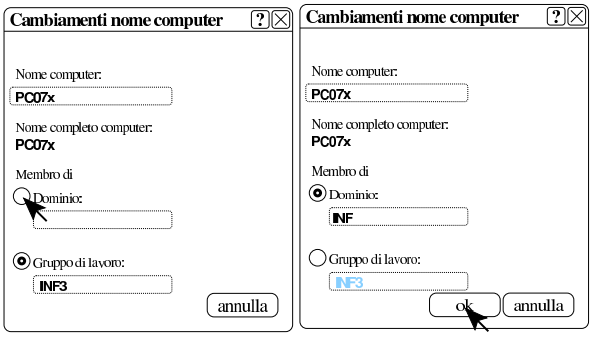
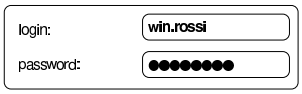


Figura u43.25. Cambiamenti nome computer: selezione del dominio e conferma.



Quando si vuole associare il dominio e confermare, occorre identificarsi in qualità di amministratore per la gestione delle utenze remote. L'utenza amministrativa in questione è la stessa usata per la connessione con PuTTY, ma con l'aggiunta del prefisso «win.».

Figura u43.26. Richiesta di identificazione per l'utente amministrativo con cui ottenere l'aggiunta del dominio.



Al termine viene richiesto di riavviare il sistema per poter rendere operative le modifiche. Al riavvio può essere scelto se utilizzare le utenze locali preesistenti o il dominio appena collegato.

Configurazione di MS-Windows 7 per utilizzare le utenze centralizzate

Per prima cosa è necessario creare due voci nel «registro di sistema», ovvero in ciò che si gestisce attraverso il programma '**`regedit`**'.

Le voci da aggiungere vanno collocate nel percorso '`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\LanmanWorkstation\parameters\`':

```
Computer
  |
  |-HKEY_CLASSES_ROOT
  |-HKEY_CURRENT_USER
  |-HKEY_LOCAL_MACHINE
  |   |
  |   '->SYSTEM
  |        |
  |        '->CurrentControlSet->services->LanmanWorkstation->parameters
  |
  |-HKEY_USERS
  '-HKEY_CURRENT_CONFIG
```

Figura u43.28. Per avviare il programma '**`regedit`**' occorre digitarne il nome nel campo di ricerca, completando alla fine con [*Invio*].

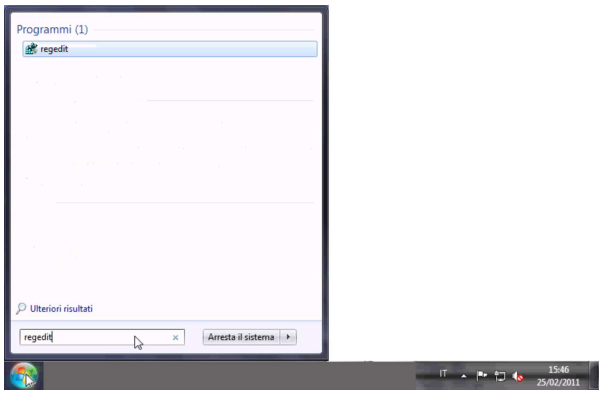
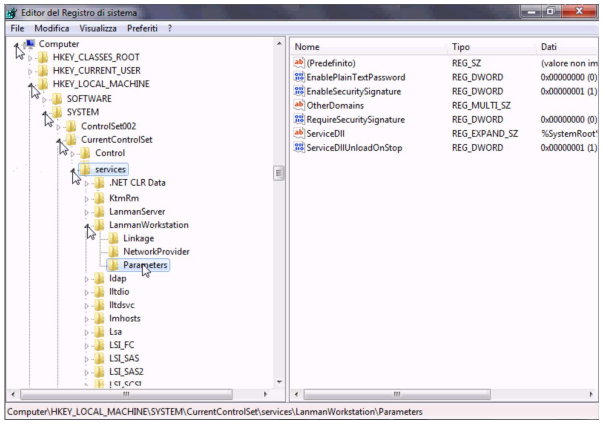


Figura u43.29. Svolgimento del percorso '`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\LanmanWorkstation\parameters\`'.



Le voci da aggiungere sono di tipo «DWORD» (nel senso di interi a 32 bit), denominate ***DomainCompatibilityMode*** e ***DNSNameResolutionRequired***. Alla prima di queste due voci si associa il valore 1, mentre alla seconda si deve lasciare il valore zero.

Figura u43.30. Creazione di una voce.

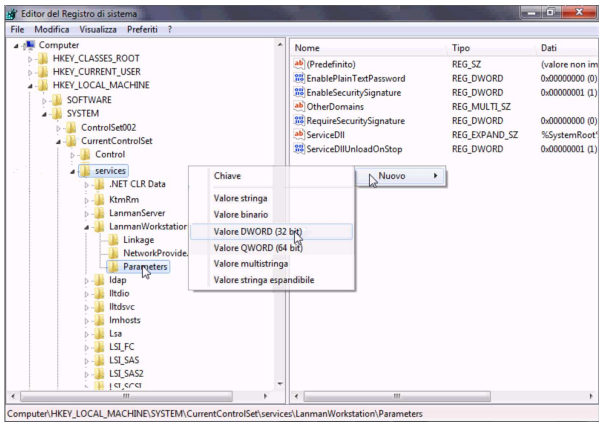


Figura u43.31. Creazione di una voce e modifica del suo contenuto.

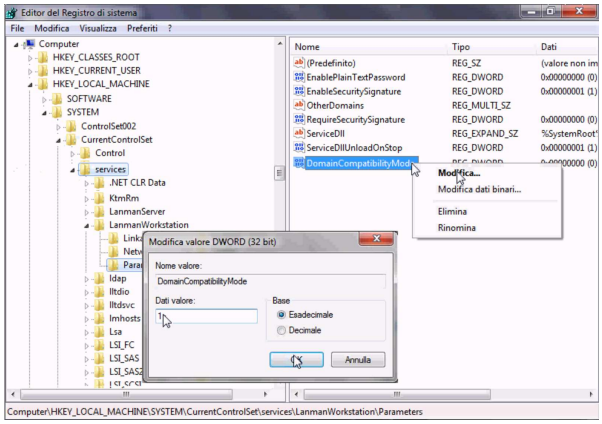
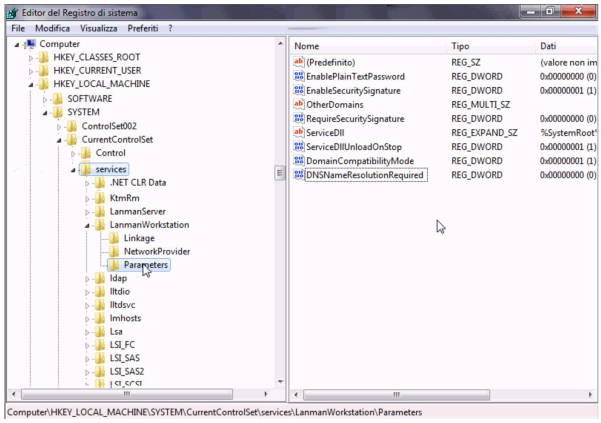


Figura u43.32. Dopo la creazione delle due voci.



Successivamente, per configurare MS-Windows 7 in modo da poter utilizzare le utenze centralizzate, è necessario associare l'elaboratore al dominio «INF». Per questo occorre abilitare l'accesso dell'elaboratore presso il servente, pertanto, dal menù di amministrazione a cui si accede con l'ausilio di PuTTY (come già descritto per la gestione delle utenze), si utilizza la voce *add machine*:

```
.--Add a new Win machine-----.
| Please insert the new Win  |
| machine name:              |
| .------------------------. |
| |                        | |
| '------------------------' |
|----------------------------|
|   <  OK   >   <Cancel>     |
'----------------------------'
```

A titolo di esempio si considera che si tratti dell'elaboratore con nome «PC29L-VAIO». Il nome va inserito usando solo lettere minuscole:

`pc29l-vaio` OK

```
.Full Win machine description--.
| Please insert the machine    |
| full description.            |
| .--------------------------. |
| |                          | |
| '--------------------------' |
|------------------------------|
|   <  OK   >   <Cancel>       |
'------------------------------'
```

È il caso di indicare il nome del laboratorio o della stanza in cui si trova:

`Laboratorio informatica` ... OK

Una volta aggiunto l'elaboratore all'elenco di quelli ammessi al servizio, si torna alla configurazione di MS-Windows 7, per l'associazione del dominio «INF»; per questo occorre agire con i privilegi dell'utente '**`Administrator`**'.

Figura u43.35. Accesso alle proprietà.

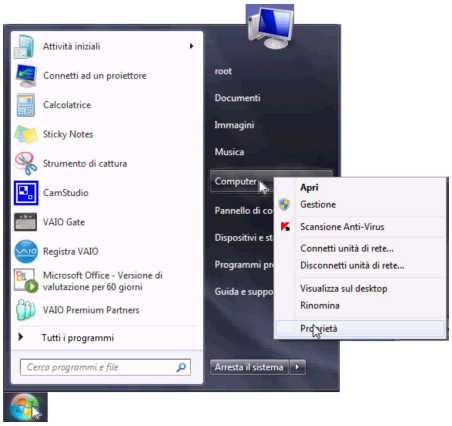


Figura u43.36. Selezione delle impostazioni avanzate.

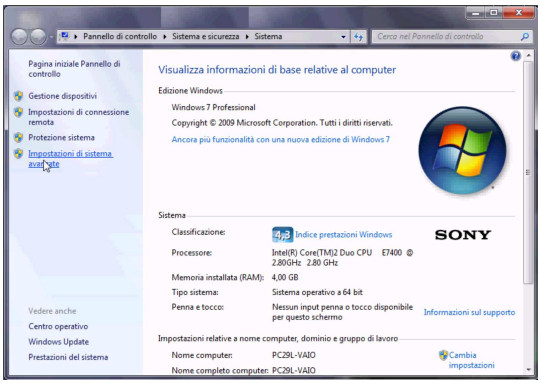


Figura u43.37. Nome dell'elaboratore.

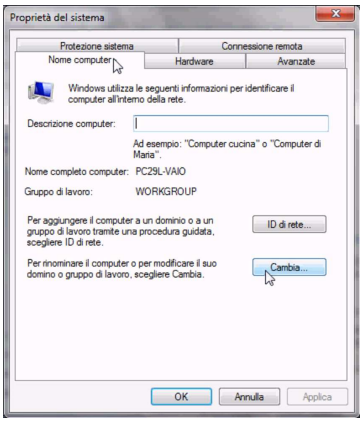
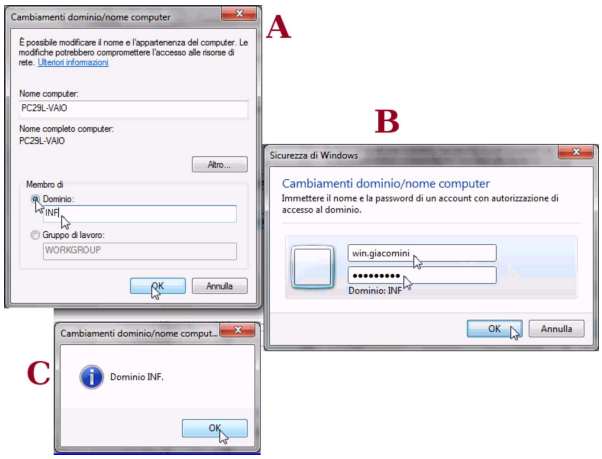
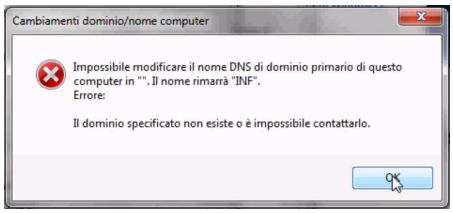


Figura u43.38. Associazione al dominio '**`INF`**' attraverso l'operato dell'amministratore «giacomini» che qui si usa come '**`win.giacomini`**'.



L'utenza amministrativa usata per completare il procedimento è la stessa che serve con PuTTY, ma con l'aggiunta del prefisso «win.».

Figura u43.39. Errore da ignorare al termine della procedura di associazione al dominio di Samba.



Configurazione per poter utilizzare anche un sistema GNU/Linux

Negli elaboratori in cui è installato MS-Windows è riavviare e ottenere un sistema GNU/Linux funzionante esclusivamente attraverso la rete.

Per ottenere questo risultato è necessario riconfigurare la sequenza di avvio degli elaboratori, in modo che appaia per prima la voce relativa all'avvio dalla rete. Di solito si tratta di abilitare la funzione di avvio dalla rete, quindi si può selezionare questa voce nella sequenza di avvio.

Avendo selezionato per primo l'avvio dalla rete, l'elaboratore presenta un menù iniziale, dove, per avviare MS-Windows non è necessario fare nulla, o al massimo si può premere semplicemente [*Invio*] per non dover attendere. Se invece si scrive il nome di una voce particolare, si ottiene l'avvio di un sistema GNU/Linux:

```
inf1      Sistema GNU/Linux per il laboratorio Informatica 1
inf2      Sistema GNU/Linux per il laboratorio Informatica 2
inf3      Sistema GNU/Linux per il laboratorio Informatica 3
inf4      Sistema GNU/Linux per il laboratorio Informatica 4
ins       Sistema GNU/Linux per la sala insegnanti
mem       Programma di controllo della memoria
```

Gli indirizzi IPv4 degli elaboratori avviati con il sistema GNU/Linux sono diversi rispetto a quando funzionano con MS-Windows. Per esempio, l'indirizzo **192.168.0.**100, usato con MS-Windows, diventa **172.17.168.**100 con GNU/Linux. In questo caso, per controllare l'accesso alla rete esterna, va usata la funzione '**`custom3`**' dal menù amministrativo.

Eliminazione delle utenze

Le utenze possono essere eliminate, assieme ai dati relativi, dal responsabile del servizio, mentre gli insegnanti che hanno la facoltà di aggiungere le utenze e di modificare le parole d'ordine non possono farlo.

Utilizzo dei laboratori MS-Windows

Premessa 219
Autenticazione 220
Accesso manuale ai dati personali da una postazione MS-Windows anonima 220
Scambio di dati tra insegnanti e studenti 222
Avvio di un sistema GNU/Linux 223
Registri elettronici 223

Gli studenti e gli insegnanti possono avvalersi di un sistema interno di gestione delle utenze e delle cartelle personali, sia presso i laboratori informatizzati con sistemi GNU/Linux, sia presso quelli con sistemi MS-Windows. Tale servizio, previa autenticazione elettronica, consente di lavorare indifferentemente presso qualunque postazione, ritrovando sempre i propri dati salvati in precedenza.

Premessa

Gli utilizzatori dei laboratori informatici basati su sistemi MS-Windows hanno la possibilità di salvare i propri dati utilizzando le cartelle personali abbinate alle utenze definite presso l'elaboratore con indirizzo IPv4 172.17.1.254, corrispondente anche all'indirizzo alternativo 192.168.0.71; inoltre, in diversi elaboratori funzionanti normalmente con un sistema MS-Windows, possono riavviare in modo da ottenere un sistema GNU/Linux (in sola lettura), con cui si accede automaticamente alle stesse cartelle personali; infine, eventualmente, tali utilizzatori possono accedere con il protocollo TELNET, attraverso il programma PuTTY, all'elaboratore 172.17.1.253, ovvero 192.168.0.81.

Il sistema di gestione delle utenze e delle cartelle personali attraverso la rete locale dell'istituto, può agevolare l'attività didattica, in quanto consente agli studenti e agli insegnanti di svolgere e ritrovare il proprio lavoro indipendentemente dal sistema operativo e dalla postazione in cui viene eseguito. Per esempio, una stessa attività didattica potrebbe essere svolta utilizzando alcune ore presso un laboratorio e altre presso un altro (purché in tutti sia disponibile il software necessario all'attività stessa), senza il problema di dover trasferire i file ogni volta; inoltre, uno stesso gruppo di studenti potrebbe svolgere certe attività con un certo insegnante e con l'uso di un sistema operativo e altre attività con un altro insegnante e con l'uso di un altro sistema. Naturalmente, tutto è perfettamente integrato con il laboratorio basato sul sistema GNU/Linux, dal momento che il servizio viene offerto precisamente da due elaboratori (*computer*) di tale laboratorio.

Va comunque osservato che ogni utente deve gestire una propria politica di copie di sicurezza, perché il servizio viene gestito con la massima cura, ma non si può mai escludere la possibilità di una perdita dei dati.

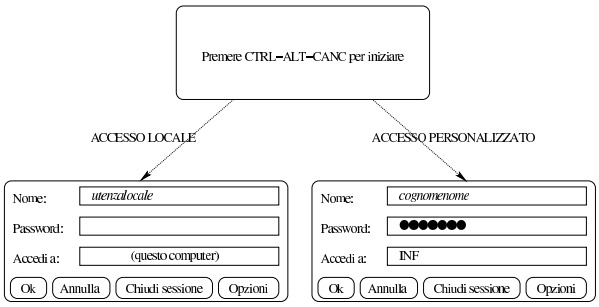
Oltre al problema della gestione dei dati attraverso la rete, vengono attuati degli accorgimenti che possono facilitare agli insegnanti il recupero delle verifiche didattiche dei propri studenti, in forma elettronica, e la pubblicazione agli stessi studenti di materiali, sempre in forma elettronica.

Alcuni insegnanti hanno la facoltà di creare le utenze ed eventualmente di cambiare la parola d'ordine agli utenti che l'hanno dimenticata. Gli insegnanti che desiderano acquisire tali privilegi, possono rivolgersi al responsabile del servizio.

Autenticazione

Presso gli elaboratori MS-Windows configurati per la gestione dei «domini», è possibile accedere con il proprio nominativo utente e con la relativa parola d'ordine, definiti a livello di istituto (e appartenenti al dominio «INF»). Presso tali elaboratori potrebbero però essere previste anche delle utenze locali, da usare in caso di emergenza o per fini particolari. Per cominciare, per accedere viene proposta la richiesta di premere la combinazione di tasti [*Ctrl Alt Canc*], quindi si ottiene la maschera per l'inserimento di nominativo-utente, parola d'ordine e contesto: se si vedono solo i primi due campi, occorre selezionare il bottone grafico `OPZIONI`.

Figura u44.1. Modalità di autenticazione: a sinistra si utilizza un'utenza locale, la quale potrebbe anche essere prima di parola d'ordine; a destra si utilizza l'utenza gestita dal dominio «INF».



Gli utenti che utilizzano l'accesso personalizzato, specificando il proprio nominativo associato al dominio «INF», trovano i propri dati personali nella risorsa di rete '`Z:\`'.

Al termine del lavoro, la sessione va chiusa, in modo che un altro utente possa autenticarsi. Se la sessione non viene chiusa e se il sistema viene lasciato in funzione, si abbandonano i propri dati personali in mano all'utilizzatore successivo.

Accesso manuale ai dati personali da una postazione MS-Windows anonima

Per accedere alle directory personali (o cartelle personali), attraverso un sistema MS-Windows che viene utilizzato in modo anonimo o comunque solo in modo locale, gli utenti devono seguire una procedura che varia in funzione della versione di tale sistema operativo. Quello che si vede negli schemi successivi è una semplificazione che dovrebbe consentire di comprendere il procedimento, adattandolo poi alla realtà del proprio sistema effettivo.

Figura u44.2. Aggiunta di una risorsa di rete.

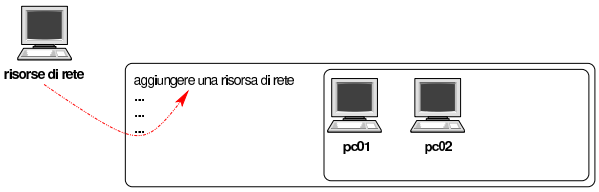


Figura u44.3. Indicazione del percorso della risorsa. L'elaboratore in cui è in funzione NLNX con il servizio Samba per la condivisione delle directory personali è raggiungibile all'indirizzo IPv4 172.17.1.254, oppure 192.168.0.71. Inoltre, l'utente ipotetico che deve collegarsi è «rossimario».

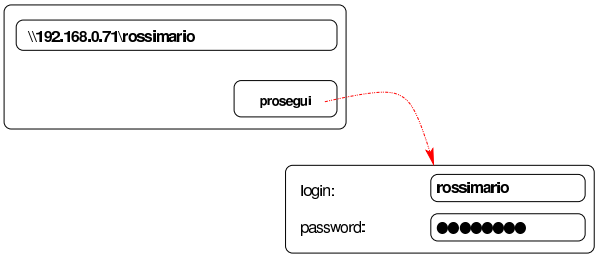
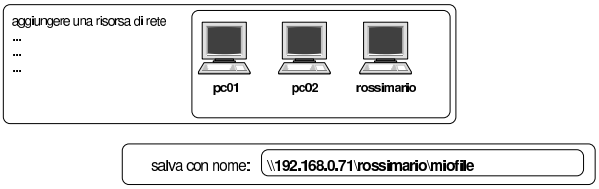


Figura u44.4. La risorsa risulta disponibile ed è possibile accedervi come se fosse un'unità a disco. Una volta collegata la risorsa, si suppone di voler salvare al suo interno un file con il nome '`miofile`'.



È molto probabile che il sistema MS-Windows chieda di memorizzare la parola d'ordine inserita: è evidente che ciò non va fatto, altrimenti un estraneo potrebbe accedere conoscendo semplicemente il nominativo-utente. Inoltre, al termine dell'utilizzo della risorsa, è necessario procedere al suo distacco, come si farebbe con un'unità rimovibile, altrimenti i dati rimarrebbero accessibili.

Le figure successive mostrano il procedimento in un sistema MS-Windows 7.

Figura u44.5. Per connettersi a una risorsa di rete è necessario selezionare la voce *Connetti unità di rete* contenuta nel menù che si ottiene premendo il tasto destro sulla voce *Computer*.

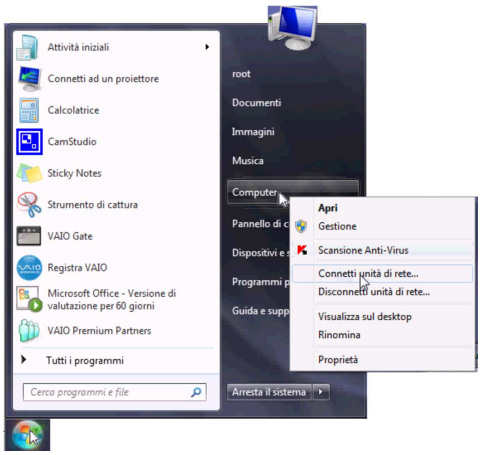


Figura u44.6. Si deve quindi specificare il percorso per raggiungere la propria cartella; in questo caso si tratta dell'elaboratore 172.17.1.254 e la cartella ha il nome '`giacominidaniele`'.

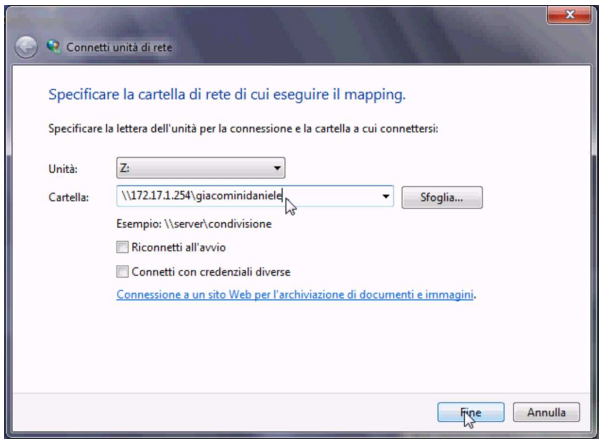


Figura u44.7. Se il percorso specificato esiste, viene richiesto di identificarsi.

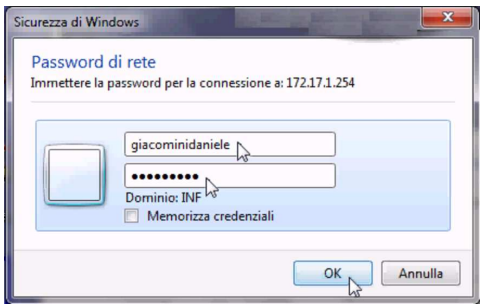
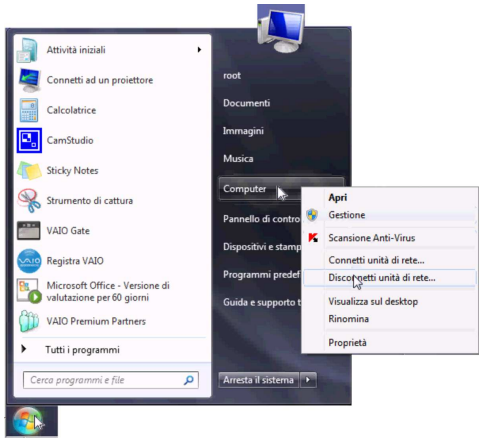


Figura u44.8. Successivamente, per chiudere la connessione con un'unità di rete occorre utilizzare nuovamente il menù che si ottiene facendo un clic con il tasto destro del mouse sopra la voce *Computer*.



Scambio di dati tra insegnanti e studenti

Le utenze create nel modo descritto dovrebbero prevedere già due sottocartelle, denominate rispettivamente '`verifiche\`' e '`strumenti\`'. Gli studenti mettono i file delle verifiche all'interno di cartelle del tipo '`verifiche\`***`insegnante`***`\`', mentre gli insegnanti mettono a disposizione dei file ai loro studenti mettendoli all'interno di cartelle del tipo '`strumenti\`***`classe`***`\`'.

Per esempio, si prenda lo studente Mario Rossi, della classe 5Ci, la cui cartella personale corrisponde alla directory '`/home/5CI1213/rossimario/`' presso l'elaboratore 192.168.0.71 (ovvero 172.17.1.254), e l'insegnante Sempronio Dicembrino, la cui directory personale corrisponde a '`/home/DOCENTI0/dicembrinosempr/`' presso lo stesso elaboratore. Lo studente Rossi prepara una verifiche per l'insegnante Dicembrino, costituita dal file '`esercizio`' che colloca nella propria cartella personale sotto '`verifiche\dicembrinosempr\`'; per converso, l'insegnante Dicembrino trova nella propria cartella personale, sotto '`verifiche\home\5CI1213\rossimario\`' il file '`esercizio`'.

Nello stesso modo, l'insegnante Dicembrino predispone un file, denominato '`modello`' e lo mette nella propria cartella sotto '`strumenti\5CI1213\`', perché gli studenti della classe 5Ci (dell'anno scolastico 2012/2013) possano utilizzarlo. Per converso, lo studente Rossi trova nella propria cartella, sotto '`strumenti\dicembrinosempr\`' il file '`modello`' predisposto dall'insegnante.

Avvio di un sistema GNU/Linux

In molti casi, gli elaboratori utilizzati normalmente con il sistema MS-Windows, possono essere avviati in modo da funzionare invece con un sistema GNU/Linux. Per ottenere questo, all'avvio è disponibile un menù, dove, in mancanza di una selezione o premendo semplicemente [*Invio*], si ottiene l'avvio di MS-Windows, mentre per ottenere un sistema GNU/Linux occorre indicare espressamente la voce relativa alla configurazione del proprio laboratorio:

```
inf1      Sistema GNU/Linux per il laboratorio Informatica 1
inf2      Sistema GNU/Linux per il laboratorio Informatica 2
inf3      Sistema GNU/Linux per il laboratorio Informatica 3
inf4      Sistema GNU/Linux per il laboratorio Informatica 4
ins       Sistema GNU/Linux per la sala insegnanti
mem       Programma di controllo della memoria
```

Una volta avviato il sistema GNU/Linux, per arrestarlo, alla fine della sessione di lavoro è disponibile un pulsante grafico denominato [`SHUTDOWN`], selezionando il quale si ottiene, per due secondi, la facoltà di scegliere se spegnere o riavviare l'elaboratore.

Registri elettronici

Il registro degli accessi degli studenti e di tutti gli altri utenti è disponibile, per controlli, presso gli indirizzi seguenti: il primo riguarda le autenticazioni provenienti da elaboratori con sistema MS-Windows, il secondo riguarda quelle relative a elaboratori con sistema GNU/Linux.

http://192.168.0.71/cgi-bin/var_log?samba/log.nanohost .

http://192.168.0.71/cgi-bin/var_log?syslog .

Tutori, pupilli e directory personali 225
La stampa 226
Pubblicazione di file attraverso la directory «~/public_html/» 226
Elenco dei tutori e dei pupilli 227
Registri 227
Amministrazione da parte dei tutori 227
Accesso normale al sistema 229
Trasferimento dati tramite il protocollo SSH 230

Presso *http://informaticalibera.net* è disponibile un servizio sperimentale di accesso remoto, attraverso il protocollo TELNET o SSH, rivolto principalmente alla didattica, basato sul sistema GNU/Linux NLNX. Il servizio distingue gli utenti in due gruppi, definiti rispettivamente ***tutori*** e ***pupilli***, dove i tutori hanno la facoltà di aggiungere e gestire le utenze dei pupilli. Per ottenere un'utenza da tutore occorre scrivere all'amministratore del servizio, descrivendo il tipo di attività che si intende svolgere: appunti2@gmail.com.

Il servizio, oltre che essere sperimentale e gratuito, viene offerto solo se effettivamente se ne presenta la richiesta. In caso di inutilizzo, il servizio viene sospeso, per risparmiare energia elettrica. Eventualmente, in caso di bisogno, può esserne richiesta la riattivazione. In ogni caso, non può esserne garantito il funzionamento e la banda è limitata per motivi economici.
Non vengono eseguite copie di sicurezza e in caso di problemi di qualunque genere, tutto potrebbe essere perduto. Le condizioni di utilizzo, molto semplici, si leggono nella pagina *http://telnet.informaticalibera.net* .

Tutori, pupilli e directory personali

Le directory personali degli utenti che rappresentano tutori e pupilli, si trovano in posizioni insolite. Per esempio, il tutore '**`rossimario`**' ha la directory personale '`/home/guestrooms/tutor/rossimario/`', mentre il pupillo '**`verdigiuseppe`**', associato al tutore '**`rossimario`**', ha la directory personale '`/home/guestrooms/pupil/rossimario/verdigiuseppe/`'. Questa organizzazione consente di individuare facilmente l'appartenenza di un pupillo al proprio tutore.

I pupilli sono utenti che operano sotto la responsabilità del proprio tutore, pertanto, al tutore viene dato un certo potere: la directory personale di ogni pupillo appartiene effettivamente all'utente corrispondente al tutore, ma al gruppo privato del pupillo stesso. Ciò significa che i permessi di accesso alla directory personale di ogni pupillo sono controllati dal tutore rispettivo. Per consentire ai pupilli di operare è necessario che i permessi di lettura, scrittura e accesso, relativi al gruppo, siano tutti disponibili, mentre ciò che si può limitare sono i permessi relativi agli altri utenti. Vengono descritti alcuni casi importanti nell'elenco seguente.

```
drwxrwxr-x 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe
```

- Questa è la situazione standard, in cui, agli altri utenti viene concesso di accedere e di leggere il contenuto della directory. In questo modo, i file prodotti dal pupillo '**`verdigiuseppe`**' sono visibili e accessibili dagli altri. Il pupillo può pubblicare dei file nella directory '`~/public_html/`'.

```
drwxrwx--x 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe
```

- Qui viene omesso il permesso di lettura della directory, in modo che gli altri utenti non possano conoscere i nomi dei file e delle altre sottodirectory. Il permesso di accesso consente comunque al

pupillo di pubblicare dei file nella directory '`~/public_html/`'.

```
drwxrwx--- 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe
```

- Qui viene omesso il permesso di lettura e di accesso alla directory, in modo che gli altri utenti non possano raccogliere alcun dato. La mancanza del permesso di accesso impedisce al pupillo di pubblicare dei file nella directory '`~/public_html/`'.

```
drwxrwxr-t 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe

drwxrwx--t 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe

drwxrwx--T 4 rossimario verdigiuseppe 4096 2012-01-01 12:00 verdigiuseppe
```

- In questi casi, risulta attivo il bit Sticky, con il quale, solo il proprietario dei file o delle sottodirectory contenuti può rimuoverli. In questo modo, per esempio, il tutore potrebbe ammettere l'accesso alla directory per tutti gli utenti, ma creare una directory '`~/public_html/`' vuota e senza permessi di scrittura per il pupillo, impedendogli in pratica di pubblicare alcunché.

Gli utenti hanno comunque la disponibilità di poco spazio su disco: circa 20 Mbyte. Questa limitazione serve per evitare l'uso del servizio per fini impropri, diversi dalla didattica e lo studio dei sistemi Unix.

La stampa

Trattandosi di un elaboratore remoto, la stampa vera e propria non è possibile; tuttavia, se si usa il comando '**`lpr`**', o '**`lp`**', passando i file attraverso lo standard input, si ottiene un file PDF nella directory '`/home/guestrooms/print/`'. Per esempio, l'utente '**`verdigiuseppe`**' vuole stampare il file '`/etc/services`':

```
$ cat /etc/services | lpr [Invio]
```

Al posto della stampa, nella directory '`/home/guestrooms/print/`' appare un file con un nome simile a '`verdigiuseppe.20121231235555.pdf`'. In pratica, il numero che appare nel nome rappresenta l'anno, il mese, il giorno, l'ora, i minuti e i secondi del momento della stampa.

I file che si trovano nella directory '`/home/guestrooms/print/`' sono visibili dall'indirizzo *http://telnet.informaticalibera.net/print/* , ma rimangono a disposizione per un tempo limitato, pari a circa un'ora. Quando scade il tempo a disposizione per lo scarico, i file vengono trasferiti nella directory '`~/print/`' del tutore rispettivo.

La directory '`/home/guestrooms/print/`' è accessibile a tutti gli utenti, i quali potrebbero creare altri file, ma non cancellare quelli di altri, come avviene con la directory temporanea '`/tmp/`'. In ogni caso, a cadenza oraria, tali file vengono trasferiti nella directory '`~/print/`' del tutore relativo.

Pubblicazione di file attraverso la directory «~/public_html/»

Gli utenti, tutori o pupilli, possono pubblicare dei file a partire dalla propria directory personale '`~/public_html/`', purché tutte le directory intermedie abbiano tutti i permessi di accesso (x) e i file siano leggibili da tutti. A questo proposito, i tutori che non vogliono consentire la pubblicazione di file ai propri pupilli, possono attivare il bit *sticky* alle directory personali dei pupilli, creando poi, al loro interno, la directory '`public_html/`', ma togliendo a questa tutti i permessi di accesso: in tal modo, i pupilli, non potendo rimuovere la directory e non potendo cambiarne i permessi, si troverebbero nell'impossibilità di farne alcunché.

```
:) rossimario@172.21.254.23:~$ cd ~verdigiuseppe [Invio]

:) rossimario@172.21.254.23:/home/guestrooms/pupil/rossimario/verdi
o+t . [Invio]
```



```
:) rossimario@172.21.254.23:/home/guestrooms/pupil/rossimario/verdi
public_html [Invio]

:) rossimario@172.21.254.23:/home/guestrooms/pupil/rossimario/verdi
a-rwx public_html [Invio]
```

L'esempio mostra il tutore '**`rossimario`**' che interviene nella directory personale del pupillo '**`verdigiuseppe`**', attribuendo prima il bit *sticky* e poi creando la directory '`public_html/`', ma senza permessi.

Quando i tutori consentono ai propri pupilli di pubblicare dei file, i tutori stessi sono comunque responsabili di quei contenuti. Per facilitare il controllo, presso la pagina *http://telnet.informaticalibera.net/tutors_and_pupils* è disponibile l'elenco dei file che ogni pupillo ha messo a disposizione, con i permessi necessari, nella pagina '`~/public_html/`'.

Elenco dei tutori e dei pupilli

Presso *http://telnet.informaticalibera.net/tutors_and_pupils* è possibile visionare l'elenco degli utenti tutori e pupilli; per la precisione, i pupilli sono raggruppati al di sotto dei tutori rispettivi. Per ogni utente appare la data e l'ora dell'ultimo accesso che ha comportato l'uso della shell, il nominativo e la descrizione dell'utente, l'elenco dei file pubblicati, incluse le stampe ancora disponibili.

Registri

Presso *http://telnet.informaticalibera.net/home_guestrooms_log* è possibile visionare i registri relativi al funzionamento dell'elaboratore che offre il servizio di accesso. I file di questi registri sono consultabili anche dall'interno del sistema, nella directory '`/home/guestrooms/log/`'.

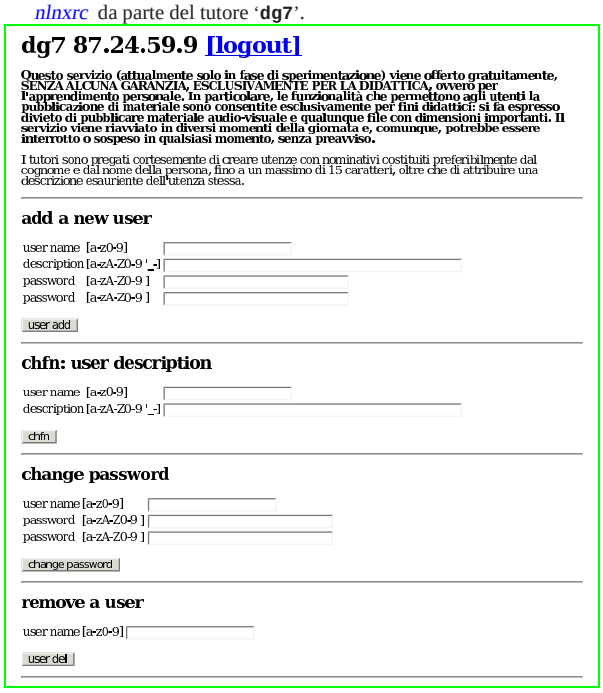
I file sono di due tipi: quelli il cui nome inizia per '`access`' e quelli che iniziano per '`syslog`'. Il primo gruppo di file raccoglie gli accessi di un certo periodo; da lì è possibile sapere quando un certo utente ha fatto un accesso al sistema; il secondo gruppo raccoglie tutte le informazioni relative al sistema operativo dell'elaboratore.

È importante ricordare che questi registri sono accessibili a tutti, attraverso la rete, anche a chi non ha un'utenza presso il sistema.

Amministrazione da parte dei tutori

I tutori sono utenti che hanno ottenuto un accesso direttamente dal gestore del sistema; a loro volta, possono creare modificare e rimuovere delle utenze da pupilli. Questa gestione amministrativa dei tutori si svolge attraverso la pagina *http://telnet.informaticalibera.net/nlnxrc* .

Figura u45.1. Accesso a *http://telnet.informaticalibera.net/nlnxrc* da parte del tutore '**dg7**'.



Il formulario che compone la pagina, include tutte le funzionalità di gestione delle utenze: creazione, modifica e cancellazione. Il formulario va compilato nei campi relativi alla funzione di proprio interesse, selezionando poi il bottone grafico che rappresenta l'azione da compiere. Per esempio, per creare un nuovo utente pupillo, si compilano i campi del gruppo *add new user*, quindi si seleziona il bottone USER ADD.

La risposta che si ottiene, selezionando un bottone grafico, consiste nello stesso formulario, con l'aggiunta dei messaggi prodotti dall'esecuzione dell'azione scelta. L'esito dell'operazione richiesta si determina dal contenuto e dal tono di tali messaggi.

Figura u45.2. Aggiunta dell'utente '**martinocalpurni**'.

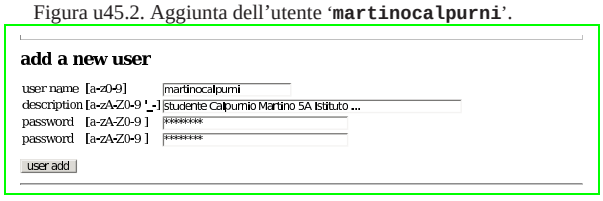
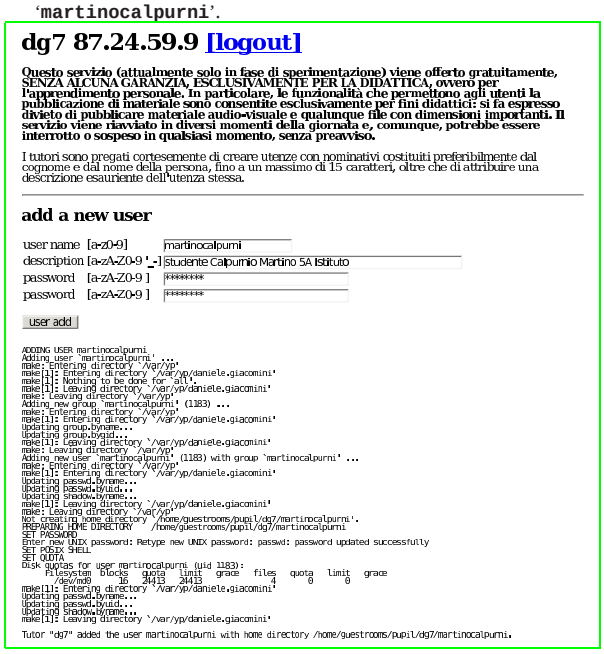


Figura u45.3. Conferma dell'aggiunta dell'utente '**martinocalpurni**'.



Quando si riceve una pagina contenente dei messaggi di conferma, come nell'esempio della figura precedente, si può procedere con altre funzioni; per esempio si può creare un altro utente. I campi di tutto il formulario vengono aggiornati con i dati ricevuti dall'ultima azione richiesta; per esempio, appena creato l'utente '**martinocalpurni**', si può subito passare alla sua cancellazione, con la sola selezione del bottone USER DEL, perché il campo relativo risulta già compilato con tale nominativo.

Accesso normale al sistema

Per accedere al sistema operativo dell'elaboratore remoto, ci si deve avvalere del protocollo TELNET. L'esempio seguente mostra l'uso del programma '**telnet**', da un terminale con sistema Unix:

```
$ telnet telnet.informaticalibera.net [Invio]

  Trying 79.137.57.90...
  Connected to telnet.informaticalibera.net.
  Escape character is '^]'.

  Linux 2.6.25.11 (79.137.57.90) (pts/1)

nlnx login: verdigiuseppe [Invio]

Password: digitazione_all'oscuro [Invio]

  This is a simple GNU/Linux distribution adapted to work
  directly from a read only file system, like a CD/DVD live
  or a USB solid-state memory.
  <synellipsis>

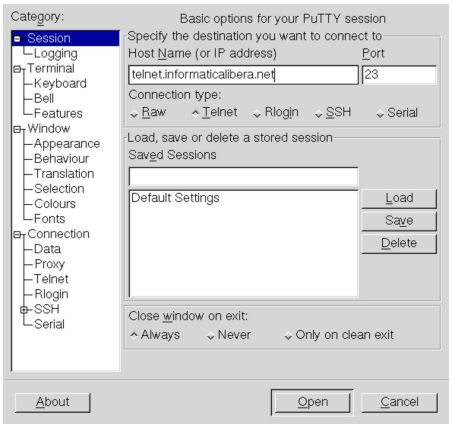
:-) verdigiuseppe@172.21.254.23:~$
```

Da questo punto si possono impartire dei comandi; al termine del lavoro, basta usare il comando '**exit**':

```
:-) verdigiuseppe@172.21.254.23:~$ exit [Invio]

  logout
  Connection closed by foreign host.
```

In alternativa, per accedere all'elaboratore remoto, si può usare il programma Putty, il quale è disponibile sia per sistemi MS-Windows, sia per sistemi GNU/Linux. All'avvio si presenta con la maschera seguente, sulla quale va configurato l'uso del protocollo TELNET:



Il comportamento successivo è lo stesso già descritto per il programma '**telnet**'.

Trasferimento dati tramite il protocollo SSH

Il servizio di accesso remoto, relativo a *http://telnet.informaticalibera.net* è rivolto alla didattica, pertanto si considera che le attività svolte non richiedano la protezione di un collegamento cifrato. Tuttavia, per il trasferimento di file, da e verso l'utenza remota, si può usare il protocollo SSH, con il comando '**scp**' o '**PSCP.EXE**'.

Il comando '**scp**' si trova nei sistemi Unix, mentre '**PSCP.EXE**' si trova nei sistemi MS-Windows (ma da usare attraverso un terminale DOS), come parte del pacchetto di Putty.

Il funzionamento dei due programmi è sostanzialmente equivalente. Vengono mostrati due esempi:

- `$ scp -P 2222 mio_file.txt` ↩
  ↪ `verdigiuseppe@telnet.informaticalibera.net:`↩
  ↪`/home/guestrooms/pupil/rossimario/verdigiuseppe` [*Invio*]

  `C:\> PSCP -P 2222 mio_file.txt` ↩
  ↪ `verdigiuseppe@telnet.informaticalibera.net:`↩
  ↪`/home/guestrooms/pupil/rossimario/verdigiuseppe` [*Invio*]

  Copia il file '`mio_file.txt`' nella directory '`/home/guestrooms/pupil/rossimario/verdigiuseppe/`' dell'elaboratore remoto, la quale è presumibilmente la directory personale dell'utente '**verdigiuseppe**', il quale è a sua volta un pupillo dell'utente '**rossimario**'.

  ```
  verdigiuseppe@telnet.informaticalibera.net's password: omissis
  [Invio]

    mio_file.txt                  100%  622KB 622.4KB/s   00:01
  ```

- `$ scp -P 2222` ↩
  ↪ `verdigiuseppe@telnet.informaticalibera.net:`↩
  ↪`/home/guestrooms/pupil/rossimario/verdigiuseppe/mio_file.txt` [*Invio*]

  `C:\> PSCP -P 2222` ↩
  ↪ `verdigiuseppe@telnet.informaticalibera.net:`↩
  ↪`/home/guestrooms/pupil/rossimario/verdigiuseppe/mio_file.txt` [*Invio*]

  Copia il file '`mio_file.txt`' dalla directory '`/home/guestrooms/pupil/rossimario/verdigiuseppe/`', presso l'elaboratore remoto, in qualità di utente '**verdigiuseppe**', nella directory corrente del sistema da cui si sta operando.

  ```
  verdigiuseppe@telnet.informaticalibera.net's password: omissis
  [Invio]

    mio_file.txt                  100%  622KB 622.4KB/s   00:01
  ```

Si può osservare l'uso dell'opzione '**-P 2222**', con la quale si richiede l'uso della porta 2222, al posto di quella predefinita per il protocollo SSH. Infatti, il servizio SSH per questo tipo di operazioni è in funzione in una porta differente rispetto allo standard.

Come funziona il servizio telnet.informaticalibera.ne

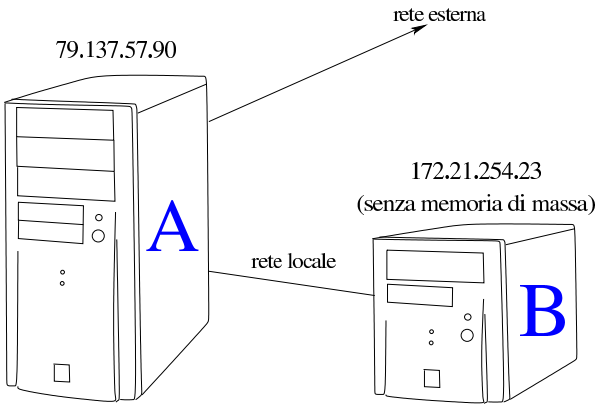
Elaboratori coinvolti 231
Ridirezione del traffico TCP 231
Avvio dalla rete dell'elaboratore «B» 232
Dati personali delle utenze privilegiate 233
Servizi via HTTP presso l'elaboratore «A» 234
Riferimenti 235

Il servizio offerto da *http://telnet.informaticalibera.net* può essere allestito facilmente utilizzando il sistema NLNX, con la dovuta configurazione. In questo capitolo si descrive in che modo è organizzato internamente il servizio *http://telnet.informaticalibera.net*, proprio per favorirne la «clonazione» in altri contesti.

Elaboratori coinvolti

Per il servizio di *http://telnet.informaticalibera.net* sono coinvolti due elaboratori, dei quali uno solo è in comunicazione con l'esterno.

Figura u46.1. Due elaboratori servono per la gestione del servizio *http://telnet.informaticalibera.net*.



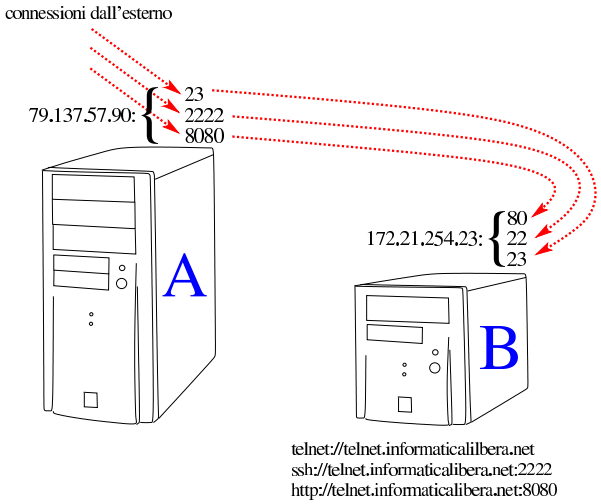
L'elaboratore che a sinistra appare indicato con la lettera «A», è un servente realizzato con NLNX, per vari tipi di servizi, in particolare funziona da firewall e da NAT, separando la rete esterna dalla rete locale. L'elaboratore a destra, indicato con la lettera «B», si trova nella rete locale, come si può vedere dall'indirizzo IPv4, il quale appartiene all'insieme riservato per le reti private. Questo secondo elaboratore è privo di unità di memorizzazione di massa (dischi, memorie solide e simili) e viene avviato dalla rete, attraverso i servizi offerti dall'elaboratore «A».

Il sistema operativo avviato dal secondo elaboratore è sempre NLNX, nella versione in sola lettura avviato dalla rete, come descritto nella sezione u28.

Ridirezione del traffico TCP

L'elaboratore «A» mette in atto tre ridirezioni di traffico TCP, per mettere in contatto l'elaboratore «B» con l'esterno: quando viene richiesta la porta 23 presso l'elaboratore «A» (TELNET), la connessione viene trasferita all'elaboratore «B», presso la stessa porta; quando viene richiesta la porta 2222 presso l'elaboratore «A», la connessione viene trasferita all'elaboratore «B», presso la porta 22 (SSH); quando viene richiesta la porta 8080 presso l'elaboratore «A», la connessione viene trasferita all'elaboratore «B», presso la porta 80 (HTTP).

Figura u46.2. Schema della ridirezione del traffico TCP verso l'elaboratore «B».



Per attuare questa ridirezione, nel file '`/etc/init.d/nlnx.network`' dell'elaboratore «A», la variabile di ambiente '**`GENERIC_FILTER`**' riceve la stringa «false», in modo da personalizzarne il funzionamento. In questo caso particolare, si trova il codice seguente:

```
#!/bin/sh
…
GENERIC_FILTER="false"
…
    elif    [ "$INTERNAL_IPV4" = "$INTERNAL_ROUTER" ]
    then
        echo "[$0] This is the internal router for the local network."
        …
        #
        # Telnet
        #
        if [ "$GENERIC_FILTER" = "true" ]
        then
            true
        else
            #
            # Telnet is redirected to the «telnet machine» and controlled.
            #
            /sbin/iptables  -t filter -A FORWARD -p tcp --dport 23 -j ACCEPT
            iptables -t nat -A PREROUTING -p tcp -s 0/0   \
                     -d $EXTERNAL_IPV4 --dport 23 -j DNAT \
                     --to-destination 172.21.254.23:23
            #
            # 8080 is redirected to 80 to the «telnet machine».
            #
            /sbin/iptables  -t filter -A FORWARD -p tcp --dport 80 -j ACCEPT
            iptables -t nat -A PREROUTING -p tcp -s 0/0     \
                     -d $EXTERNAL_IPV4 --dport 8080 -j DNAT \
                     --to-destination 172.21.254.23:80
            #
            # SSH
            #
            /sbin/iptables  -t filter -A FORWARD -p tcp --dport 22 -j ACCEPT
            #
            iptables -t nat -A PREROUTING -p tcp -s 0/0     \
                     -d $EXTERNAL_IPV4 --dport 2222 -j DNAT \
                     --to-destination 172.21.254.23:22
        fi
        …
```

Avvio dalla rete dell'elaboratore «B»

Come già accennato nel capitolo, l'elaboratore «B» si avvia dalla rete, utilizzando un sistema NLNX. Per questo NLNX prevede che si possa procedere soltanto nell'ambito di una rete locale, condividendo le directory personali e la gestione delle utenze dall'elaboratore che offre già il servizio di avvio del sistema. Pertanto, l'elaboratore «A» deve offrire tutto ciò che serve al funzionamento dell'elaboratore «B», ma deve essere in grado di distinguere tra le utenze privilegiate, le quali possono accedere direttamente all'elaboratore «A», dalla rete locale o dall'esterno tramite SSH, rispetto alle utenze di tutori e pupilli, le quali possono accedere attraverso TELNET, per essere però ridirette all'elaboratore «B».

In questa fase si vuole considerare solo la configurazione del DHCP e del servizio PXE, per l'avvio dalla rete, fornendo i parametri di avvio del kernel necessari alla configurazione corretta dell'elaboratore «B».

Prima di tutto, è necessario che il servizio DHCP offerto dall'elaboratore «A» associ univocamente l'interfaccia di rete dell'elaboratore «B» all'indirizzo IPv4 stabilito, ovvero 172.21.254.23. Pertanto, con il comando '**`nlnxrc dhcp_server config`**' è stato annotato un abbinamento simile a quello successivo:

```
172.21.254.23  00:30:1B:AB:FB:39
```

Nel file '`/etc/dhcp3/dhcpd.conf`' ciò si traduce nella direttiva seguente:

```
host 172.21.254.23 { hardware ethernet 00:30:1B:AB:FB:39;
                     fixed-address 172.21.254.23; }
```

Quindi si passa alla definizione del file '`/var/lib/tftpboot/pxelinux/pxelinux.cfg/AC15FE17`'. il nome '`AC15FE17`' rappresenta in esadecimale l'indirizzo IPv4 172.21.254.23, pertanto questo file diventa la configurazione personalizzata dell'elaboratore con tale indirizzo:

```
DISPLAY pxelinux.msg
TIMEOUT 100
PROMPT 1
DEFAULT telnet

label telnet
  kernel vmlinuz
  append n_boot=net root=/dev/ram0 ro init=/linuxrc ↩
↪initrd=nlnxrd.img ramdisk_size=30720 n_no_way_out=1 ↩
↪n_xorg_conf=0 n_reboot=05:30+13:30 n_max_pages=30 ↩
↪n_lpr_directory=/home/guestrooms/print
```

Si possono vedere delle opzioni di avvio insolite, specifiche di NLNX, il cui significato viene descritto nell'elenco seguente:

- '**`n_no_way_out=1`**'

  Il sistema avviato con questa opzione, configura un firewall interno in cui disabilita le connessioni verso l'esterno, rivolte a servizi non vitali.

- '**`n_reboot=05:30+13:30`**'

  Viene innescato il riavvio dell'elaboratore alle 5:30 e alle 13:30 di ogni giorno.

- '**`n_lpr_directory=/home/guestrooms/print`**'

  Si stabilisce che la stampa deve essere ridiretta verso file PDF, da creare nella directory '`/home/guestrooms/print/`'.

Dati personali delle utenze privilegiate

Il servizio in questione serve a offrire a un gran numero di persone, anche minorenni, accesso a un elaboratore. Una cosa che si deve cercare di evitare è di permettere l'accesso, anche erroneamente, ai dati personali di utenti che utilizzano i servizi dell'elaboratore «A» al di fuori del conteso didattico di studio che riguarda invece gli utenti tutori e pupilli. Per cercare di ovviare a questo problema, le utenze di tutori e pupilli hanno una directory personale che discende dal percorso '`/home/guestrooms/`', mentre le utenze si trovano in percorsi diversi, anche se discendenti da '`/home/`'.

L'elaboratore «A», oltre alla gestione delle utenze, gestisce la condivisione delle directory personali che si articolano tutte da '`/home/`'. Nel file '`/etc/exports`' dell'elaboratore «A» si fa in modo che l'elaboratore «B» possa innestare solo il percorso '`/home/guestrooms/`', rimanendo così all'oscuro di tutto il resto del contenuto della directory '`/home/`':

```
/home/guestrooms 172.21.0.0/16(async,rw,root_squash,nohide,no_subtree_check)
/home            172.21.11.0/24(async,rw,root_squash,nohide,no_subtree_check)
/opt/nlnx        172.21.0.0/16(ro,no_root_squash,sync,nohide,subtree_check)
```

Come si vede dalla porzione mostrata del file '`/etc/exports`', tutti gli elaboratori con indirizzi del tipo 172.21.∗.∗, possono innestare la directory '`/home/guestrooms/`', mentre la directory '`/home/`'

può essere innestata direttamente solo da una parte che ha indirizzi del tipo 172.21.*.*.

In pratica, il sistema NLNX avviato dalla rete, prima tenta di innestare la directory '`/home/`', poi, non riuscendoci, prova con '`/home/guestrooms/`'.

Servizi via HTTP presso l'elaboratore «A»

Presso l'elaboratore «A» si accede attraverso il protocollo HTTP, per la gestione delle utenze e per funzioni simili. Per questo scopo sono previsti alcuni script e programmi nella distribuzione di NLNX, collocati nella directory '`/etc/script.cgi/`', ma il servente HTTP (Mathopd) va configurato di conseguenza. L'estratto seguente del file di configurazione '`/etc/mathopd.conf`' mostra le porzioni significative al riguardo:

```
…
Server {
    Port 80
    Address 0.0.0.0
    …
    Virtual {
        Host telnet.informaticalibera.net
        #
        Control {
            Alias /nlnxrc
            Location /etc/script.cgi/nlnxrc.cgi
            Realm nlnxrc.cgi
            UserFile /etc/nlnx/CGI_PASSWD
            Specials {
                CGI { nlnxrc.cgi }
            }
        }
        Control {
            Alias /home_guestrooms_log
            Location /etc/script.cgi/home_guestrooms_log
            Specials {
                CGI { home_guestrooms_log }
            }
        }
        Control {
            Alias /tutors_and_pupils
            Location /etc/script.cgi/tutors_and_pupils.cgi
            Specials {
                CGI { tutors_and_pupils.cgi }
            }
        }
        Control {
            Alias /print
            Location /home/guestrooms/print
            AutoIndexCommand /etc/script/index.cgi
            Specials {
                CGI { index.cgi }
            }
        }
        Control {
            Alias /home/guestrooms/print
            Location /home/guestrooms/print
            AutoIndexCommand /etc/script/index.cgi
            Specials {
                CGI { index.cgi }
            }
        }
    }
}
```

Il programma '**`nlnxrc.cgi`**' acquisisce i privilegi dell'utente '**`root`**' (è configurato come SUID-root) e quindi avvia lo script '**`nlnxrc.su.cgi`**'. Questo script è responsabile della gestione delle utenze e ci si avvale del file '`/etc/nlnx/CGI_PASSWD`' per riconoscere gli utenti tutori.

Lo script '**`home_guestrooms_log`**' non richiede privilegi particolari e si occupa di visualizzare il contenuto della directory '`/home/guestrooms/log/`', assieme ai suoi file.

Il programma '**`tutors_and_pupils.cgi`**' acquisisce i privilegi dell'utente '**`root`**' per avviare lo script '**`tutors_and_pupils.su.cgi`**', il quale produce l'elenco dei tutori e dei pupilli, assieme a tutti i riferimenti ai documenti pubblicati dagli utenti rispettivi. Lo script richiede i privilegi dell'utenza amministrativa, per poter leggere alcuni file contenuti nelle directory personali degli utenti.

Riferimenti

- *Download PuTTY*
  http://www.putty.org

Parte xvi

Installazione e avvio di un sistema GNU/Linux

«

Installazione di una distribuzione Slackware 239

Organizzazione dei dischetti di avvio 239
Avvio dai dischetti e preparazione all'installazione 240
Avvio della procedura di installazione 241
Configurazione del sistema 244
Conclusione 245
Riferimenti 245

LILO: introduzione 247

Organizzazione essenziale 247
Installazione del meccanismo di caricamento del sistema operativo 247
LILO su un disco differente 250
Boot prompt 251
Riferimenti 252

Configurazione di LILO più in dettaglio (omesso) 253

Direttive di configurazione globale 254
Direttive utilizzabili globalmente e anche nelle sezioni specifiche 256
Sezioni delle voci di avvio 258
Esempi 259

Installazione di una distribuzione Slackware

Organizzazione dei dischetti di avvio 239
Avvio dai dischetti e preparazione all'installazione 240
Avvio della procedura di installazione 241
Configurazione della partizione di scambio 242
Selezione delle partizioni di tipo Linux-nativa 242
Selezione della fonte della distribuzione GNU/Linux 243
Selezione dei gruppi di pacchetti da installare 243
Installazione dei pacchetti e del kernel 244
Configurazione del sistema 244
Dischetto di avvio 244
Configurazione e installazione del sistema di avvio 245
Ora locale 245
Conclusione 245
Riferimenti 245

La distribuzione GNU/Linux Slackware è ancora attiva nonostante l'età e la sua forma piuttosto spartana. L'installazione di questa distribuzione richiede una certa confidenza con i sistemi Unix, o almeno una certa disponibilità; in questo capitolo si vuole descrivere in modo non troppo dettagliato il procedimento, per dare una visione generale della logica che c'è sotto. La lettura di questo capitolo, permette di comprendere anche l'utilizzo di altre procedure di installazione più semplici.

Nel capitolo si fa riferimento a una distribuzione Slackware degli anni 1990, installata in un elaboratore che dispone di un lettore CD/DVD di tipo ATAPI per l'installazione, usando per l'avvio dei dischetti.

In una distribuzione attuale è improbabile l'uso di dischetti di avvio, a causa delle dimensioni del kernel. Al loro posto si usano piuttosto CD o DVD autoavviabili.

Organizzazione dei dischetti di avvio

Per l'installazione di una distribuzione Slackware degli anni 1990, si può preparare una coppia di dischetti, dove il primo, definito *boot disk*, serve per l'avvio, mentre il secondo, il *root disk*, contiene un sistema minimo utile per l'installazione o per risolvere dei problemi in caso di emergenza.

I dischetti che si intendono utilizzare devono essere stati inizializzati, ma soprattutto devono essere privi di difetti. Anche se l'inizializzazione dei dischetti non riporta errori o settori danneggiati, ciò non basta a garantire che questi siano perfetti. Durante il loro utilizzo, nella fase di installazione, potrebbero mostrarsi delle segnalazioni di errore, oppure il sistema potrebbe bloccarsi durante l'avvio. In tali casi, conviene tentare nuovamente utilizzando dischetti differenti.

I dischetti si preparano a partire dai file-immagine, **senza decomprimerli**, anche se i nomi dei file in questione possono avere estensioni particolari, come '`.gz`'.

Una caratteristica molto importante della distribuzione Slackware è quella di avere predisposto una grande quantità di file-immagine per il dischetti di avvio, ognuna con un kernel diverso, più adatto per questo o quel dispositivo fisico. In tal modo non si fa uso di moduli (o almeno non troppi) e, al massimo, si possono inserire dei parametri di avvio se l'hardware non viene rilevato in modo automatico. In generale, l'immagine contenuta nel file '`bootdsks.144/bare.i`' è quella più adatta alle situazioni «normali», nel caso in cui si disponga di unità di memorizzazione ATA, compresi i lettori CD-ROM ATAPI. A ogni modo, nella directory che contiene i file delle immagini dei dischetti di avvio, sono contenuti dei file di testo

che descrivono in modo chiaro le caratteristiche dei kernel contenuti nelle stesse.

Per quanto riguarda la scelta del dischetto contenente l'immagine del sistema operativo minimo, per l'installazione occorre scegliere il file-immagine '`rootdsks/color.gz`', a meno che si stia tentando un'installazione particolare, per la quale potrebbe essere disponibile un file specifico con un altro nome. Anche in questo caso sono disponibili dei file di testo che guidano alla scelta.

Avvio dai dischetti e preparazione all'installazione

Dopo aver preparato uno spazio sufficiente a contenere il sistema GNU/Linux nel disco fisso e dopo aver preparato la coppia di dischetti necessaria a cominciare, si può avviare il proprio elaboratore a partire dal dischetto di avvio, che è quello contenente il kernel.

Dopo la conclusione della fase diagnostica attivata dal BIOS, viene letto e caricato il dischetto di avvio e quindi visualizzato un messaggio introduttivo. Alla fine appare un invito particolare che permette di inserire istruzioni speciali da comunicare al kernel. Alcuni tipi di dispositivo richiedono l'indicazione di un'istruzione di questo genere perché il kernel possa riconoscerli. In questa fase potrebbe essere necessario indicare qualcosa per fare in modo che venga riconosciuto un lettore CD-ROM speciale, o un disco fisso SCSI. Per conoscere il modo corretto di dare queste istruzioni, occorre leggere la premessa che viene visualizzata.

```
  DON'T SWITCH ANY DISKS YET! This prompt is just for entering extra parameters.
  If you don't need to enter any parameters, hit ENTER to continue.

boot:
```

Di solito basta premere [*Invio*] senza indicare alcun parametro particolare:

```
boot: [Invio]
```

Viene quindi caricato il kernel contenuto nel dischetto e durante questa fase vengono visualizzate una serie di informazioni diagnostiche sui componenti hardware riconosciuti o meno. È da questi messaggi che si può capire se le unità di memorizzazione e di connessione alla rete che si vogliono utilizzare, sono riconosciute, così come dovrebbero, in base al tipo di file-immagine scelto per l'avvio.

Alla fine, appare il messaggio seguente che invita a sostituire il dischetto di avvio con quello contenente il sistema minimo (viene utilizzato il dischetto ottenuto dal file '`color.gz`').

```
  VFS: Insert root floppy disk to be loaded into ramdisk and press ENTER
```

Appena è stato sostituito il dischetto si deve premere [*Invio*].

```
[Invio]

  - You will need one or more partitions of type 'Linux native' prepared.  It is
    also recommended that you create a swap partition (type 'Linux swap') prior
    to installation.  For more information, run 'setup' and read the help file.

  - If you're having problems that you think might be related to low memory (this
    is possible on machines with 8 or less megabytes of system memory), you can
    try activating a swap partition before you run setup.  After making a swap
    partition (type 82) with cfdisk or fdisk, activate it like this:
      mkswap /dev/<partition> ; swapon /dev/<partition>

  - Once you have prepared the disk partitions for Linux, type 'setup' to begin
    the installation process.

  - If you do not have a color monitor, type:  TERM=vt100
    before you start 'setup'.

  You may now login as 'root'.
```

Dopo una serie di altre informazioni che riassumono le operazioni da compiere in casi particolari di installazione, appare il messaggio seguente che invita ad accedere al mini sistema appena avviato, utilizzando il nominativo-utente '**`root`**', per il quale non viene richiesta alcuna parola d'ordine.

```
  You may now login as "root"

slakware login: root [Invio]
```

Dopo aver inserito il nominativo-utente '**`root`**' (e dopo aver premuto [*Invio*]), si ottiene l'invito della shell, rappresentato dal simbolo seguente:

```
#
```

Il messaggio introduttivo che precede la richiesta dell'inserimento del nominativo-utente, dà una serie di indicazioni sulle cose da fare prima di avviare lo script '**`setup`**' che inizia la procedura di installazione. È necessario utilizzare subito '**`fdisk`**', per definire le partizioni del disco fisso; eventualmente, è anche possibile attivare manualmente l'utilizzo di partizioni di scambio per estendere la memoria virtuale.

Avvio della procedura di installazione

Dopo la preparazione delle partizioni, il lavoro più importante è già fatto e si è pronti per iniziare l'installazione vera e propria del sistema GNU/Linux attraverso lo script '**`setup`**'. Prima però, può essere il caso di porgere attenzione allo schermo: se si dispone di un monitor monocromatico, conviene modificare il contenuto della variabile '**`TERM`**', per esempio nel modo seguente:

```
# TERM=vt100 [Invio]
```

Per avviare lo script di installazione, non servono opzioni:

```
# setup [Invio]
```

Si ottiene la visualizzazione del menù generale della procedura di installazione, come si vede nella figura u47.5.

Figura u47.5. Menù generale della procedura di installazione.

```
.------------------------ Slackware Linux Setup ------------------
| Welcome to Slackware Linux Setup.
| Select an option below using the UP/DOWN keys and SPACE or ENTER
| Alternate keys may also be used: '+', '-', and TAB.
| .-----------------------------------------------------------------.
| |   HELP       Read the Slackware Setup HELP file                 |
| |   KEYMAP     Remap your keyboard if you're not using a US one   |
| |   ADDSWAP    Set up your swap partition(s)                      |
| |   TARGET     Set up your target partitions                      |
| |   SOURCE     Select source media                                |
| |   SELECT     Select categories of software to install           |
| |   INSTALL    Install selected software                          |
| |   CONFIGURE  Reconfigure your Linux system                      |
| |   EXIT       Exit Slackware Linux Setup                         |
| `-----------------------------------------------------------------'
|-----------------------------------------------------------------.
|                      <  OK  >      <Cancel>                     |
`-----------------------------------------------------------------'
```

Per l'interazione con l'utilizzatore, lo script '**`setup`**' fa uso del programma '**`dialog`**', con il quale si generano facilmente delle finestre di dialogo, anche se solo per lo schermo a caratteri. Appare generalmente un cursore, o una zona evidenziata, che rappresenta un'opzione attiva o semplicemente la posizione corrente a cui possono fare riferimento i comandi della tastiera. Si possono utilizzare le tecniche consuete per interagire con questo programma: i tasti freccia spostano il cursore nella direzione della freccia; il tasto [*Tab*] permette di passare da un elemento all'altro; per selezionare un pulsante grafico occorre posizionare il cursore sul pulsante stesso e quindi premere [*Invio*]; per selezionare una voce da una lista, occorre posizionare il cursore sulla voce desiderata e successivamente si deve premere [*Invio*]; per selezionare o deselezionare una casella di selezione (*check box*), si deve posizionare il cursore sulla casella desiderata e premere la [*barra spaziatrice*].

Questo script può essere utilizzato anche all'interno di un sistema GNU/Linux già installato e funzionante. In tal caso, il menù cambia leggermente e alcune opzioni hanno un comportamento un po' diverso.

La sequenza delle voci del menù iniziale suggerisce l'ordine in cui dovrebbero essere svolte le operazioni. La prima cosa da fare dovrebbe essere la lettura della guida, corrispondente alla voce *HELP*, per essere informati sulle ultime novità e sulle cose a cui si deve prestare attenzione; quindi è opportuno configurare subito la tastiera con l'aiuto della voce *KEYMAP*.

Configurazione della partizione di scambio

Si suppone che la partizione di scambio sia già stata creata prima di avviare la procedura di installazione. In questa fase è importante definirne l'utilizzo e la sua attivazione. Si fa questo selezionando la voce *ADDSWAP* del menù iniziale (vi si porta sopra il cursore e quindi si seleziona il pulsante grafico OK).

La procedura mostra l'elenco delle partizioni di scambio ritrovate, per esempio quello seguente:

```
Slackware setup has detected a swap partition:

   Device Boot   Begin    Start      End   Blocks   Id  System
/dev/hda2            83       83      148    33264   82  Linux swap

Do you wish to install this as your swap partition?
```

La risposta a questa domanda è un sì:

YES

La procedura si premura di avvisare che occorre fare attenzione a non inizializzare e nemmeno attivare una partizione di scambio già attivata, con o senza l'ausilio della procedura stessa:

```
IMPORTANT NOTE: if you have already made any of your swap
partitions active (using the swapon command), then you
should not allow Setup to use mkswap on your swap partitions,
because it may corrupt memory pages that are currently
swapped out. Instead, you will have to make shure that your
swap partitions have been prepared (with mkswap) before they
will work. You might want to do this to any inactive swap
partitions before you reboot.
```

OK

La procedura di installazione, dopo l'avvertimento appena riportato, chiede se si intende inizializzare le partizioni attraverso l'utilizzo di '**mkswap**':

```
Do you want Setup to use mkswap on your swap partitions?
```

YES

```
Your swapspace has been configured. This information will
be added to your /etc/fstab:

/dev/hda2       swap         swap         defaults   1   1
```

OK

Al termine, la procedura di installazione propone di proseguire consigliando la prossima fase, quella corrispondente alla voce *TARGET* del menù iniziale.

Selezione delle partizioni di tipo Linux-nativa

Dal menù generale della procedura di installazione si può selezionare la voce *TARGET*: si ottiene la visualizzazione di tutte le partizioni di tipo Linux-nativa ovvero quelle corrispondenti al codice 83_{16}. Nel caso mostrato dall'esempio, si tratta di un'unica partizione primaria:

```
Please select a partition from the following list to use for
your root (/) Linux partition.

        /dev/hda3  Linux native, 441504K
        ----  (add none, continue with setup)
        ----  (add none, continue with setup)
        ----  (add none, continue with setup)
```

La partizione principale (*root*) è quella che serve ad avviare il sistema. Nella maggior parte dei casi è anche l'unica. L'elenco di partizioni, in questo caso si tratta di un elenco di un'unica partizione, si comporta come un menù: si tratta di selezionare la prima e unica partizione portandoci sopra il cursore con l'aiuto dei tasti [*freccia-su*] o [*freccia-giù*] e selezionando il pulsante grafico OK.

Quindi, la procedura di installazione propone di inizializzare o controllare la partizione:

```
Format   Quick format with no bad block checking
Check    Slow format that checks for bad blocks
No       No, do not format this partition
```

La prima delle scelte corrisponde all'esecuzione di '**mke2fs**', la seconda all'esecuzione di '**mke2fs -c**', la terza non esegue alcuna inizializzazione. Se la partizione è già stata inizializzata in precedenza, magari in modo manuale, non occorre ripetere l'operazione, ma

se viene ripetuta non comporta inconvenienti. La scelta si fa spostando il cursore sulla voce desiderata e selezionando il pulsante grafico OK.

Quando si vuole scomporre il file system in più partizioni, è necessario collegarle insieme, specificando i vari punti di innesto. Per questo motivo, se la procedura di installazione rivela la presenza di più partizioni di tipo Linux-nativa (83_{16}), dopo l'indicazione della partizione principale richiede la selezione delle altre partizioni con l'indicazione di una directory di destinazione. In pratica, quella partizione va a contenere tutti i dati a partire da quella directory.

Selezione della fonte della distribuzione GNU/Linux

Attraverso questa fase, si informa la procedura di installazione della posizione in cui si trovano i file della distribuzione GNU/Linux da utilizzare per l'installazione. Dal menù generale si seleziona l'opzione *SOURCE*. Viene proposto un menù di scelta dell'origine:

```
1  Install from a Slackware CD-ROM
2  Install from a hard drive partition
3  Install via NFS
4  Install from a pre-mounted directory
5  Install from floppy disks (A and N series only)
```

Come indicato all'inizio del capitolo, si fa riferimento solo all'installazione da CD-ROM.

1 OK

Viene proposta la possibilità di cercare automaticamente l'unità in cui è stato inserito il CD-ROM, oppure di indicare dove sia:

```
auto    Scan for the CD-ROM drive automatically
manual  Manually select CD-ROM device
```

Volendo scegliere la voce *manual* dall'elenco proposto, si ottiene un altro elenco contenente i nomi dei file di dispositivo riferiti a tutti i lettori che potrebbero esistere:

```
custom          Type in the CD-ROM device to use
/dev/hdb        CD-ROM slave on first IDE bus
/dev/hda        CD-ROM master on first IDE bus (unlikely)
/dev/hdc        CD-ROM master on second IDE bus
/dev/hdd        CD-ROM slave on second IDE bus
   …
/dev/scd0       First SCSI CD-ROM drive
/dev/scd1       Second SCSI CD-ROM drive
   …
/dev/pcd0       First parallel port ATAPI CD
/dev/pcd1       Second parallel port ATAPI CD
   …
/dev/aztcd      Non-IDE Aztech CD-ROM
/dev/cdu535     Sony CDU-535 CD-ROM
/dev/gscd       Non-IDE GoldStar CD-ROM
/dev/sonycd     Sony CDU-31a CD-ROM
   …
```

Se si suppone che il lettore sia collocato nella terza unità ATA, si deve selezionare */dev/hdc*. Dopo la selezione del dispositivo, se il CD-ROM della distribuzione viene individuato effettivamente, si passa all'indicazione del modo in cui deve essere fatta l'installazione:

```
slakware        Normal installation to hard drive (best performanc
slaktest        Link /usr -> /cdrom/live/usr to run mostly from CD
custom          Install from a custom directory
help            Read the installation method help file
```

In condizioni normali si seleziona la modalità *slakware*, che comporta l'installazione completa nel disco fisso.

Selezione dei gruppi di pacchetti da installare

Attraverso questa fase, a cui si accede dalla voce *SELECT* del menù generale, si identificano le categorie delle applicazioni GNU/Linux da installare nel disco fisso. Viene proposto un elenco nel quale alcune categorie raccomandate appaiono già selezionate:

```
[X] A    Base Linux system
[X] AP   Various Applications that do not need X
[X] D    Program Development (C, C++, Lisp, Perl, etc.)
[X] E    GNU Emacs
[X] F    FAQ lists, HOWTO documentation
[X] K    Linux kernel source
[X] N    Networking (TCP/IP, UUCP, Mail, News)
[X] T    TeX typesetting software
[X] TCL  Tcl/Tk script languages
[X] X    XFree86 X Window System
[X] XAP  X Applications
[ ] XD   X Server development kit
[X] XV   XView (OpenLook Window Manager, apps)
[X] Y    Games (that do not require X)
```

Per selezionare o deselezionare una categoria, è possibile portarvi sopra il cursore e usare la [*barra spaziatrice*]; una volta segnati i gruppi che si intendono installare si conferma con il pulsante grafico [OK]. Naturalmente, finché non si ha una buona esperienza, conviene lasciare le scelte predefinite.

Installazione dei pacchetti e del kernel

Dopo la selezione delle categorie si può passare all'installazione vera e propria: direttamente dopo la selezione o a partire dal menù generale selezionando la voce *INSTALL*. Viene quindi visualizzato un altro menù che permette di scegliere il modo con cui si vuole procedere all'installazione dei vari pacchetti all'interno delle categorie prescelte:

```
full            Install everything (up to 386 MB of software)
newbie          Use verbose prompting (and follow tagfiles)
menu            Choose groups of packages from interactive menus
expert          Choose individual packages from interactive menus
custom          Use custom tagfiles in the package directories
tagpath         Use tagfiles in the subdirectories of a custom pat
help            Read the prompt mode help file
```

La scelta più comoda, almeno per gli utilizzatori normali, è la voce *full*. A questo punto inizia l'installazione dei pacchetti, al termine della quale viene richiesto espressamente di specificare quale kernel deve essere installato:

```
bootdisk        Use the kernel from the installation bootdisk
cdrom           Use a kernel from the Slackware CD
floppy          Install a zImage or bzImage from a DOS floppy
skip            Skip this menu and use the default /vmlinuz
```

Probabilmente, la scelta migliore è proprio la prima, quella corrispondente alla voce *bootdisk*, tenendo conto che successivamente conviene ricompilare il kernel in base alle proprie esigenze specifiche. Se si opta per la voce *bootdisk*, viene richiesto l'inserimento del dischetto in questione.

Configurazione del sistema

Al termine dell'installazione dei pacchetti prescelti, la procedura di installazione propone di iniziare la fase della configurazione del sistema. Si può passare alla fase di configurazione anche utilizzando l'opzione *CONFIGURE* del menù generale. Eventualmente, in caso di ripensamenti, la configurazione può essere ripetuta, saltando l'indicazione degli elementi che si ritiene siano configurati correttamente.

La sequenza successiva delle richieste fatte dalla procedura, dipende da cosa è stato installato. Nelle sezioni seguenti vengono mostrati solo alcuni punti importanti.

Dischetto di avvio

La prima fase è quella che prevede la preparazione di un dischetto da usare per l'avviamento del sistema in caso di emergenza. In pratica viene utilizzato il kernel prescelto (o l'ultimo se si è tentato di installarne più di uno) copiandolo in un dischetto. Conviene rispondere affermativamente alla proposta della procedura di installazione. Si tratta quindi di togliere l'ultimo dischetto utilizzato dall'unità (ammesso che ce ne sia ancora uno inserito) e inserire un dischetto inizializzato precedentemente. Si ottiene un elenco di possibilità:

```
format          format floppy disk in /dev/fd0
simple          make simple vmlinuz > /dev/fd0 bootdisk
lilo            make lilo bootdisk
continue        leave bootdisk menu and continue with the configura
```


Come si vede, è possibile inizializzare il dischetto prima di utilizzarlo. In generale, per realizzare un dischetto di avvio è meglio selezionare la voce *lilo*. Quindi, viene richiesto di inserire il dischetto e di confermare.

Configurazione e installazione del sistema di avvio

Inizia quindi una fase piuttosto delicata: quella dell'impostazione del sistema che si occupa di eseguire l'avvio del kernel Linux.

Per poter avviare il sistema, si deve modificare il *Master boot record*, o MBR, di conseguenza, se prima esisteva un altro sistema operativo che si avviava autonomamente, dopo questa modifica non si può avviare più, se non per mezzo del nuovo sistema di avvio. Se però questo sistema di avvio viene installato o configurato male, allora non si può avviare né GNU/Linux e nemmeno gli altri sistemi operativi eventuali.

```
simple  Try to install LILO automatically
expert  Use expert lilo.conf setup menu
skip    Do not install LILO
```

La scelta iniziale che viene proposta permette di indicare se installare o meno il sistema di avvio; in generale dovrebbe essere conveniente selezionare la voce *simple*.

```
MBR     Install to Master Boot Record
Root    Install to superblock (which must be made bootable)
Floppy  Install to a formatted floppy in /dev/fd0 (A:)
```

Le alternative successive sono abbastanza chiare: la voce *MBR* si riferisce alla possibilità di installare il sistema di avvio alterando il settore iniziale del disco, per controllare direttamente l'avvio di tutti i sistemi operativi; la voce *Root* permette di alterare solo il primo settore di una partizione, che deve essere resa avviabile in qualche altro modo (per mezzo di un altro programma); la voce *Floppy* permette di non toccare alcunché e di fare una prova con un dischetto (che non contiene il kernel, ma solo un settore di avvio). In condizioni normali, se non si teme di commettere errori, è meglio selezionare la voce *MBR*; in alternativa, la scelta più prudente è quella di usare un dischetto.

Ora locale

L'ultimo elemento della configurazione che vale la pena di prendere in considerazione è la definizione dell'ora locale. Viene proposto un elenco lunghissimo di fusi orari. Per identificarli sono state usate le città più importanti, di solito le capitali:

```
…
Europe/Prague
Europe/Riga
Europe/Rome
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
…
```

Nel caso dell'Italia si deve selezionare la voce *Europe/Rome*.

Conclusione

Al termine della configurazione, riappare il menù generale della procedura di installazione, dal quale, finalmente, si può selezionare l'uscita con la voce *EXIT*. Quello che si ottiene è nuovamente l'invito della shell, dal quale si può arrestare il sistema nel modo tradizionale.

```
# shutdown -h now [Invio]
```

Riferimenti

- *Slackware Linux*
  http://www.slackware.com/

LILO: introduzione

Organizzazione essenziale 247
Installazione del meccanismo di caricamento del sistema operativo 247
Configurazione 248
Installazione del sistema di avvio 250
LILO su un disco differente 250
Boot prompt 251
Riferimenti 252

LILO [1] è una procedura per il caricamento di GNU/Linux negli elaboratori con architettura x86. Permette di avviare anche altri sistemi operativi eventualmente residenti nello stesso elaboratore in cui si usa GNU/Linux. In questa sezione si vedono solo alcuni aspetti del suo funzionamento, quelli che dovrebbero bastare nella maggior parte delle situazioni. Nel capitolo u49 viene descritta con maggiore dettaglio la sua configurazione, ma per un approfondimento sul suo funzionamento conviene consultare la documentazione che accompagna questa procedura: la pagina di manuale *lilo(8)*, quanto contenuto nella directory '`/usr/share/doc/lilo/`' e il *BootPrompt HOWTO*.

Organizzazione essenziale

La procedura LILO è composta essenzialmente da:

- la directory '`/boot/`' e dal suo contenuto;
- l'eseguibile '**`lilo`**';
- il file di configurazione '`/etc/lilo.conf`'.

La directory '`/boot/`' contiene i file utilizzati per effettuare l'avvio del sistema: sia per avviare GNU/Linux, sia per altri sistemi operativi eventuali. Può contenere anche il file del kernel, o più file di kernel differenti, quando per questo non si usa semplicemente la directory radice. Più precisamente, contiene almeno i file seguenti:

- '`boot.b`';
- '`map`' che viene creato da LILO;
- '`boot.`***`n`***', dove l'estensione è un numero esadecimale, che viene creato da LILO e contiene il settore di avvio dell'unità rappresentata dal numero stesso (non si tratta necessariamente di un solo file);
- il kernel se non risiede già nella directory radice.

Nella tabella u48.1 sono elencati i codici esadecimali corrispondenti ad alcuni dispositivi per le unità di memorizzazione.

Tabella u48.1. Elenco dei codici esadecimali dei dispositivi di alcune unità di memorizzazione.

Dispositivo	Codice	Dispositivo	Codice	Dispositivo	Codice
`/dev/fd0`	200_{16}	`/dev/fd1`	201_{16}		
`/dev/hda`	300_{16}	`/dev/hda1`	301_{16}	`/dev/hda2`	302_{16}
`/dev/hdb`	340_{16}	`/dev/hdb1`	341_{16}	`/dev/hdb2`	342_{16}
`/dev/sda`	800_{16}	`/dev/sda1`	801_{16}	`/dev/sda2`	802_{16}
`/dev/sdb`	810_{16}	`/dev/sdb1`	811_{16}	`/dev/sdb2`	812_{16}

Installazione del meccanismo di caricamento del sistema operativo

L'installazione del meccanismo di caricamento del sistema operativo avviene modificando il contenuto di uno di questi settori:

- MBR o *Master boot record*;
- il primo settore di una partizione;
- il primo settore di un dischetto.

Nel primo caso, LILO ha il controllo su tutti i sistemi operativi per il loro caricamento; nel secondo, LILO dipende da un sistema di avviamento di un altro sistema operativo che, a sua volta, passa a LILO il controllo quando ciò viene richiesto; nel terzo caso si utilizza un dischetto in modo da non alterare il sistema di avvio già presente.

L'installazione avviene per mezzo dell'eseguibile '**`lilo`**', che a sua volta si basa sulla configurazione stabilita attraverso '`/etc/lilo.conf`'. Ogni volta che si cambia qualcosa all'interno della directory '`/boot/`', o si modifica, o si sposta il file del kernel, è necessario ripetere l'installazione attraverso l'eseguibile '**`lilo`**'.

Configurazione

Il file di configurazione utilizzato da LILO per installare il sistema di avvio è '`/etc/lilo.conf`'. Si tratta di una sorta di script contenente solo assegnamenti a variabili. Ne viene descritto il funzionamento in modo sommario partendo da un esempio in cui si ha un solo disco fisso, dove la prima partizione è riservata al Dos e la seconda a un sistema GNU/Linux. L'esempio permette di avviare GNU/Linux e il Dos selezionando una tra le parole '**`linux`**' o '**`dos`**' al momento dell'avvio. Il simbolo '**`#`**' rappresenta l'inizio di un commento che viene ignorato.

```
# Prima parte generale
boot=/dev/hda
prompt
timeout=50

# Caricamento di Linux
image=/boot/vmlinuz
    label=linux
    root=/dev/hda2
    read-only

# Caricamento del Dos
other=/dev/hda1
    label=dos
    table=/dev/hda
```

Segue la descrizione delle direttive che appaiono nell'esempio.

- '**`boot=/dev/hda`**'

  Nella prima parte viene specificato che il settore di avvio deve essere collocato nel primo disco ATA, di conseguenza nell'MBR. Se invece viene indicata una partizione specifica, allora si tratta del primo settore di quella partizione (per esempio: '**`boot=/dev/hda2`**'). Volendo si potrebbe indicare anche un'unità per i dischetti, in modo da installare tale settore di avvio in quel dischetto (per esempio: '**`boot=/dev/fd0`**').

- '**`prompt`**'

  Si tratta di un'opzione (una variabile booleana) la cui presenza fa sì che all'atto del caricamento venga richiesto di inserire il nome del sistema che si desidera avviare (per la precisione, la parola chiave che vi fa riferimento).

- '**`timeout=50`**'

  Dopo 50 decimi di secondo (cinque secondi), senza che sia stato selezionato alcunché, viene avviato il sistema predefinito (in questo caso '**`linux`**').

- '**`image=/boot/vmlinuz`**'

  Inizia la definizione di un kernel da avviare: '`/boot/vmlinuz`'.

Si tratta del file che si trova nel file system in funzione nel momento in cui si avvia l'eseguibile '**`lilo`**'. Questo particolare potrebbe sembrare ovvio, ma non è sempre così. Se si vuole preparare un sistema di avvio per un sistema GNU/Linux residente in un'altra partizione (magari un dischetto), si vuole forse fare riferimento a un kernel che si trova lì (nel dischetto). La cosa potrebbe non essere tanto intuitiva e viene descritta più avanti.

- '**`label=linux`**'

  Definisce il nome utilizzato per fare riferimento a questo kernel. Potrebbe essere qualunque cosa, in questo caso il nome '**`linux`**' è utile per ricordare che si tratta dell'avvio di quel sistema operativo.

- '**`root=/dev/hda2`**'

  Indica la partizione da utilizzare come file system principale (*root*).

- '**`read-only`**'

  La presenza di questa opzione fa sì che la partizione specificata venga innestata inizialmente in sola lettura, in modo da permettere al kernel di eseguire un controllo prima di avviare il resto del sistema. Al termine del controllo, la partizione viene reinnestata regolarmente in lettura e scrittura, ma questo per opera della procedura di inizializzazione del sistema.

- '**`other=/dev/hda1`**'

  Inizia la definizione dell'avvio di un altro sistema operativo, per il quale non è LILO a prendersi cura dell'avvio del kernel, ma un altro settore di avvio. In questo caso il settore di avvio deve trovarsi all'inizio della partizione '`/dev/hda1`'.

- '**`label=dos`**'

  Definisce il nome utilizzato per fare riferimento a questo sistema operativo. La parola '**`dos`**' è utile per ricordare che si tratta dell'avvio di quel sistema operativo.

- '**`table=/dev/hda`**'

  Specifica il file di dispositivo che si riferisce all'unità che contiene l'indicazione della tabella delle partizioni. In effetti, questa è contenuta nella parte iniziale del disco fisso, quindi si fa riferimento all'intera unità '`/dev/hda`'.

Volendo, è possibile avviare lo stesso file system con kernel differenti a seconda delle necessità. In tal caso si possono aggiungere al file '`/etc/lilo.conf`' altri blocchetti come quello seguente:

```
# Caricamento di Linux con un kernel sperimentale
image=/boot/vmlinuz-prova
    label=prova
    root=/dev/hda2
    read-only
```

Se si vuole la possibilità di utilizzare come file system principale una partizione diversa da quella normale, magari per fare delle prove, o per qualunque altro motivo, si può indicare una voce alternativa come quando si vuole avviare con diversi kernel possibili.

```
# Caricamento di una partizione alternativa in un disco SATA
image=/boot/vmlinuz
    label=extra
    root=/dev/sda3
    read-only
```

Quello che conta è comprendere che il sistema di avvio resta nella directory '`/boot/`' e senza il disco che la contiene, i file system in '`/dev/hda2`' o '`/dev/sda3`' non possono essere innestati. Inoltre, senza '`/dev/hda`' (in questi esempi), non si avvierebbe alcunché. Per comprendere meglio il problema, si pensi a questo esempio:

- GNU/Linux sia avviato e stia utilizzando la partizione '`/dev/hda2`' come file system principale;
- la directory '`/boot/`' sia vuota e sia stata utilizzata per innestare un dischetto corrispondente al dispositivo '`/dev/fd0`';

- la directory radice del dischetto corrisponda esattamente a '`/boot/`';
- il dischetto contenga i file già visti, necessari per l'avvio (il kernel, '`boot.b`', '`map`', ecc.);
- il file '`/etc/lilo.conf`' sia come quello visto sopra, per cui il settore di avvio si deve trovare nell'MBR del primo disco fisso ('`/dev/hda`').

In questo modo, se si esegue '**`lilo`**', viene creato un settore di avvio nell'MBR di '`/dev/hda`' che fa riferimento ai file di avvio (kernel incluso) contenuti nel dischetto. Cioè, senza quel dischetto (proprio quello), il sistema non potrebbe avviarsi. Questo problema viene rivisto più avanti dove viene spiegato come costruire un dischetto contenente sia un settore di avvio, sia il kernel e i file di LILO.

Alle volte è necessario informare il kernel di qualche particolarità dell'hardware installato. In tal caso si utilizza la variabile '**append**' alla quale si assegna la stringa necessaria. Nell'esempio seguente si invia la stringa '**`cdu31a=0x340,0`**' necessaria per poter attivare un vecchio lettore CD-ROM Sony.

```
# Caricamento di Linux con l'attivazione del CD-ROM
image=/boot/vmlinuz
    label=sony
    root=/dev/hda2
    append="cdu31a=0x340,0"
    read-only
```

Installazione del sistema di avvio

L'eseguibile '**`lilo`**' permette di installare il sistema di avvio basato sulla procedura LILO. Per farlo, legge il contenuto del file '`/etc/lilo.conf`' o di quello indicato attraverso l'opzione '**`-C`**'.

```
lilo [opzioni]
```

Tabella u48.6. Alcune opzioni.

Opzione	Descrizione
`-C` ***file_di_configurazione***	Permette di indicare un file di configurazione differente rispetto al solito '`/etc/lilo.conf`'.
`-r` ***directory_di_partenza***	Permette di definire una pseudo directory radice in modo da poter utilizzare quanto contenuto in un dischetto o in un altro disco innestato da qualche parte.

Segue la descrizione di alcuni esempi.

- `# lilo -C ./mia.conf` [*Invio*]

  Installa il sistema di avvio utilizzando la configurazione del file '`mia.conf`' contenuto nella directory corrente.

- `# lilo -r /mnt/floppy` [*Invio*]

  Utilizza la configurazione del file '`/mnt/floppy/etc/lilo.conf`', facendo riferimento (probabilmente) ai file contenuti in '`/mnt/floppy/boot/`', utilizzando i file di dispositivo in '`/mnt/floppy/dev/`'.

LILO su un disco differente

LILO parte dal presupposto che si stia operando sempre all'interno del file system attivo nel momento in cui si avvia l'eseguibile '**`lilo`**'. Si potrebbe pensare che per fare in modo di sistemare l'avvio su un altro disco, come un dischetto o un'altra unità rimovibile, si debba agire semplicemente sulla direttiva '**`boot=`*dispositivo***'; ma questo non basta. Si deve utilizzare l'opzione '**`-r`**' per fare riferimento a una pseudo directory radice, a partire dalla quale LILO deve trovare tutto quello che gli serve, compreso il file di configurazione.

Di seguito viene mostrato l'esempio della preparazione di un dischetto contenente il kernel avviato da LILO, in modo completamente indipendente dal file system attivo nel momento in cui lo si realizza, con una configurazione simile a quella mostrata in precedenza, nella sezione i48.2.1.

1. All'interno di un dischetto inizializzato e contenente un file system Second-extended (Ext2) si riproduce tutto quello che serve a LILO per definire il sistema di avvio. Si tratta della directory '`boot/`' contenente gli stessi file della stessa directory appartenente al file system generale, insieme al kernel; della directory '`etc/`' con il file '`lilo.conf`'; della directory '`dev/`' con i file di dispositivo corrispondenti alle unità di memorizzazione cui si fa riferimento. Si suppone di avere innestato il dischetto utilizzando la directory '`/mnt/floppy/`' come punto di innesto.

   `# fdformat /dev/fd0u1440` [*Invio*]

   `# mkfs.ext2 /dev/fd0` [*Invio*]

   `# mount -t ext2 /dev/fd0 /mnt/floppy` [*Invio*]

   `# cp -dpR /boot /mnt/floppy` [*Invio*]

   `# mkdir /mnt/floppy/etc` [*Invio*]

   `# cp /etc/lilo.conf /mnt/floppy/etc/lilo.conf` [*Invio*]

   `# mkdir /mnt/floppy/dev` [*Invio*]

   `# cd /mnt/floppy/dev/` [*Invio*]

   `# /dev/MAKEDEV fd0 fd1 hda hdb hdc hdd sda sdb sdc sdd` [*Invio*]

2. Il file '`/mnt/floppy/etc/lilo.conf`' viene modificato in modo da fare riferimento al dispositivo '`/dev/fd0`'.

   ```
   boot=/dev/fd0
   ```

3. Si utilizza l'eseguibile '**`lilo`**' con l'opzione '**`-r`**' in modo da fargli usare i file nel dischetto e non quelli contenuti nel file system principale.

   `# lilo -r /mnt/floppy` [*Invio*]

Il problema può presentarsi anche in modo inverso, quando si avvia il sistema attraverso dischetti di emergenza e si vuole sistemare l'avvio di GNU/Linux attraverso il disco fisso. La partizione principale del disco fisso potrebbe essere innestata nel sistema di emergenza, per esempio in corrispondenza della directory '`/mnt/`', mentre per il resto non dovrebbe essere necessario preoccuparsi d'altro, a parte la versione di LILO presente nel dischetto, che deve essere compatibile con i file di avvio del disco fisso.

`# lilo -r /mnt` [*Invio*]

Boot prompt

Subito dopo la prima fase dell'avvio del sistema, quella gestita da LILO, prima dell'avvio vero e proprio del kernel, in presenza di determinate condizioni viene visualizzato un invito particolare a inserire delle opzioni: il *boot prompt*. Questo appare:

- se è stata indicata l'istruzione '**`prompt`**' nel file '`/etc/lilo.conf`';
- se viene premuto il tasto [*Maiuscole*], oppure [*Ctrl*], oppure [*Alt*];
- se il tasto [*Fissamaiuscole*] oppure [*BlocScorr*] risultano inseriti.

Il *boot prompt*, ovvero l'invito dell'avvio, ha l'aspetto seguente:

```
boot:
```

Normalmente si utilizza la riga di comando di avvio per indicare il nome di una configurazione particolare. In altri casi è il mezzo per specificare un'opzione che per qualche motivo non è attiva automaticamente e si vuole che LILO la passi al kernel.

La digitazione all'interno di questa riga di comando è abbastanza intuitiva: per cancellare si possono usare i tasti [*Backspace*], [*Canc*] e le combinazioni [*Ctrl u*] e [*Ctrl x*]. Eventualmente, si può ottenere

un elenco delle configurazioni, riferite a diverse voci del file '`/etc/lilo.conf`', attraverso la pressione del tasto [*Tab*]. Si conferma con il tasto [*Invio*]. Il vero problema è la tastiera: si deve considerare che la disposizione dei tasti è quella statunitense.

La sintassi di quanto si può inserire attraverso la riga di comando è la seguente:

```
[configurazione [opzione...]]
```

Se si preme semplicemente [*Invio*] viene avviata la configurazione predefinita, altrimenti è obbligatorio l'inserimento del nome di questa, seguita eventualmente da altre opzioni.

I vari argomenti inseriti attraverso la riga di comando (il nome della configurazione e le altre opzioni eventuali) sono separati tra loro attraverso uno spazio. Per questo, un argomento non può contenere spazi.

Nella sezione 6.8 vengono descritti alcuni tipi di parametri che possono essere inseriti in una riga di comando di avvio. Per una descrizione più ampia conviene consultare il *BootPrompt HOWTO*.

Riferimenti

- Werner Almesberger, *LILO Generic boot loader for Linux - User's guide*

[1] **LILO** licenza speciale senza vincoli particolari

Configurazione di LILO più in dettaglio (omesso)

Direttive di configurazione globale 254
Direttive utilizzabili globalmente e anche nelle sezioni specifiche 256
Sezioni delle voci di avvio 258
Esempi 259

LILO è uno dei sistemi di avvio di kernel Linux e di altri sistemi operativi, specifico per gli elaboratori di architettura x86. Il suo file di configurazione è '`/etc/lilo.conf`'.

Solitamente, il file di configurazione viene creato in modo predefinito già in fase di installazione, utilizzando opzioni generiche.

Nella sostanza le direttive di configurazione hanno la forma di assegnamenti a variabili, intese come opzioni che hanno un ruolo nella fase di avvio del sistema. A parte il caso delle righe bianche e di quelle vuote, che vengono ignorate, oltre alla possibilità di indicare dei commenti preceduti dal simbolo '`#`', si usa la sintassi seguente:

```
nome=valore_assegnato
```

```
nome="valore_assegnato"
```

```
opzione_booleana
```

In particolare:

- ogni direttiva deve essere disposta su una riga propria;
- a seconda del contesto, i valori assegnati possono essere sensibili alla differenza tra maiuscole e minuscole;
- è ammissibile l'uso di uno spazio (<*SP*>), prima e dopo il simbolo '`=`' che rappresenta l'assegnamento, ma in generale si preferisce ometterlo;
- se si deve assegnare una stringa contenente uno o più spazi, occorre racchiuderla tra virgolette;
- alcune direttive rappresentano un'opzione booleana, per cui è sufficiente annotarne il nome senza alcun assegnamento, per indicare implicitamente l'abilitazione dell'opzione relativa;
- gli assegnamenti che non si possono ricondurre a direttive di configurazione, vengono intesi come assegnamenti a variabili di ambiente che poi sono passate al processo iniziale, tali e quali, rispettando anche l'uso delle lettere maiuscole o minuscole.

Le direttive di configurazione sono organizzate in sezioni: quelle della parte iniziale rappresentano la configurazione generale, mentre le sezioni specificano le particolarità delle voci che si possono selezionare nel momento dell'avvio del sistema operativo.

Tabella u49.1. Organizzazione delle direttive di configurazione di LILO.

Nome direttiva	Sezione globale	Sezione '`image`'	Sezione '`other`'
`backup=`***file***	Sì		
`force-backup=`***file***	Sì		
`boot=`***file_di_dispositivo***	Sì		
`compact`	Sì		
`default=`***riferimento_alla_sezione_predefinita***	Sì		
`delay=`***decimi_di_secondo***	Sì		
`fix-table`	Sì		

Nome direttiva	Sezione globale	Sezione ‘**image**’	Sezione ‘**other**’
`ignore-table`	Sì		
`install=`***file***	Sì		
`keytable=`***file***	Sì		
`map=`***file***	Sì		
`message=`***file***	Sì		
`nowarn`	Sì		
`prompt`	Sì		
`serial=`***porta*** `[`, ***velocità*** `[`***parità*** `[`***n-bit*** `]]]`	Sì		
`timeout=`***decimi_di_secondo***	Sì		
`verbose=`***n***	Sì		
`append=`***parametri_di_avvio_del_kernel***	Sì	Sì	
`initrd=`***file***	Sì	Sì	
`read-only`	Sì	Sì	
`read-write`	Sì	Sì	
`root=`***file***	Sì	Sì	
`vga=`{`normal`|`extended`|`ask`|***n***}	Sì	Sì	
`lock`	Sì	Sì	
`password=`***parola_d'ordine***	Sì	Sì	Sì
`restricted`	Sì	Sì	
`single-key`	Sì	Sì	
`label=`***nome***		Sì	Sì
`alias=`***nome***		Sì	Sì
`loader=`***file***			Sì
`map-drive=`***codice_virtuale*** `to=`***codice_reale***			Sì
`table=`***file_di_dispositivo***			Sì

Direttive di configurazione globale

Le direttive che appaiono all'inizio del file di configurazione, prima della dichiarazione delle sezioni specifiche, riguardano tutte le sezioni sottostanti. Implicitamente appartengono alla sezione globale che non viene dichiarata espressamente. Nel seguito vengono descritte alcune di queste.

- `backup=`***file***

  `force-backup=`***file***

  La prima delle due direttive, fa sì che nel momento in cui si installa il nuovo settore di avvio, venga fatta una copia di quello vecchio nel file specificato, a meno che il file in questione ci sia già, nel qual caso la copia non viene rifatta. In alternativa, la seconda direttiva non tiene conto dell'esistenza o meno del file, che eventualmente viene sovrascritto.

- `boot=`***file_di_dispositivo***

  Indica il nome del file di dispositivo nel quale installare il settore di avvio. In generale si tratta del file di dispositivo corrispondente a tutto il primo disco, ‘`/dev/hda`’, altrimenti, specie se si tratta

di una partizione, significa che deve essere poi un altro sistema di avvio a prendersi carico dell'avvio di questo settore particolare.

- `compact`

  Cerca di riunire le richieste di lettura relative a settori adiacenti in un'unica operazione, allo scopo di ridurre il tempo necessario a caricare il sistema operativo. L'uso di questa direttiva è particolarmente utile nella realizzazione di dischetti di avvio.

  Questa direttiva è generalmente incompatibile con la direttiva ‘**LINEAR**’, che qui non viene descritta.

- `default=`***riferimento_alla_sezione_predefinita***

  Permette di definire quale voce selezionare in modo predefinito, tra quelle disponibili, in mancanza di una scelta precisa da parte dell'utente. Il nome che viene assegnato si riferisce a quanto dichiarato all'interno delle sezioni con la direttiva ‘**image=***nome*’.

- `delay=`***decimi_di_secondo***

  Permette di specificare un ritardo, espresso in decimi di secondo, prima di avviare il sistema. Potrebbe essere necessario in alcune situazioni particolari, per dare il tempo a qualche componente fisica dell'elaboratore di inizializzarsi. In particolare, assume già un valore predefinito quando si utilizza la direttiva ‘**serial**’ per attivare l'uso di un terminale attraverso la porta seriale.

- `fix-table`

  Questa opzione booleana, consente la correzione automatica della tabella delle partizioni, all'inizio delle partizioni stesse, nel caso queste non corrispondano allo standard normale. Si intuisce che questa facoltà possa creare dei disguidi se nel disco sono installati altri sistemi operativi con le loro convenzioni particolari.

- `ignore-table`

  Con questa opzione booleana si fa in modo che vengano ignorate eventuali anomalie nella tabelle delle partizioni.

- `install=`***file***

  Con questa direttiva si specifica esplicitamente il nome del file contenente il settore di avvio da installare. Se non si indica questa direttiva, viene usato in modo predefinito il file ‘`/boot/boot.b`’.

- `keytable=`***file***

  Questa direttiva stabilisce una rimappatura della tastiera secondo la codifica riportata nel file indicato. Il file in questione deve essere generato appositamente, tenendo conto della mappa di partenza (quella del BIOS) e di quella di destinazione.

  Per ottenere questo file, si utilizza un programma che fa parte del pacchetto che compone LILO: può trattarsi di ‘**keytab-lilo.pl**’ o di ‘**keytab-lilo**’. Questo utilizza le mappe di definizione della tastiera di un sistema GNU/Linux normale, per generare ciò che serve. L'esempio seguente si riferisce al caso in cui, dalla solita tastiera inglese si passi alla disposizione italiana dei tasti:

```
# keytab-lilo /usr/share/keymaps/i386/qwerty/us.kmap.gz ↩
↪             /usr/share/keymaps/i386/qwerty/it.kmap.gz ↩
↪             > mappa_tastiera.lilo [Invio]
```

  Nell'esempio, il file che si ottiene è ‘`mappa_tastiera.lilo`’.

- `map=`***file***

Specifica la posizione e il nome del file contenente la mappa necessaria per raggiungere il kernel e altre informazioni indispensabili all'avvio. Se non si indica esplicitamente tale direttiva, viene creato e usato il file '`/boot/map`' in modo predefinito.

- `message=`***file***

Indica un file di testo contenente un messaggio che deve essere visualizzato all'avvio, prima dell'invito di LILO. La lunghezza massima del testo è di 65535 byte; in particolare, il carattere *<FF>* (che si ottiene normalmente con la combinazione [*Ctrl l*]), genera una ripulitura dello schermo.

È importante sottolineare che lo spostamento o la modifica di questo file richiede la ricostruzione del file di mappa, ovvero '`/boot/map`'.

- `nowarn`

Disabilita l'emissione di messaggi di avvertimento.

- `prompt`

Richiede la comparsa dell'invito. Di solito si usa questa direttiva assieme a '**`timeout`**', per fissare un tempo massimo oltre il quale viene selezionata automaticamente la voce predefinita.

- `serial=`***porta*`[`*`,velocità`*`[`*`parità`*`[`*`n-bit`*`]]]`

Abilita l'interazione attraverso una porta seriale:

  – ***porta*** indica il numero della porta seriale, dove lo zero corrisponde alla prima, ovvero '`/dev/ttyS0`';
  – ***velocità*** si esprime in bit/s (bps) e si riferisce alla velocità di comunicazione della porta, dove i valori ammissibili sono 110, 300, 1200, 2400, 4800, 9600, 19200 e 38400, mentre il valore predefinito è 2400;
  – ***parità*** rappresenta il tipo di parità usata dalla linea seriale, espresso attraverso la lettera '**`n`**' (nessuna parità), la lettera '**`e`**' (pari), oppure la lettera '**`o`**' (dispari);
  – ***n-bit*** rappresenta la dimensione dei caratteri trasmessi e sono ammissibili solo i valori 7 e 8, tenendo conto che in modo predefinito si intendono 8 bit se non si usa alcuna parità, altrimenti si intendono 7 bit.

A titolo di esempio, la direttiva '**`serial=1,2400n8`**' fa riferimento alla seconda porta seriale, che viene inizializzata per una connessione a 2400 bit/s, senza parità, con caratteri di 8 bit. In pratica, queste sono anche le impostazioni predefinite, per cui sarebbe stato sufficiente usare la direttiva abbreviata '**`serial=1`**'.

Si osservi che se si utilizza la direttiva '**`serial`**', si stabilisce implicitamente un ritardo di due secondi, attraverso la direttiva '**`delay=20`**'.

- `timeout=`***decimi_di_secondo***

Questa direttiva stabilisce un tempo di attesa, espresso in decimi di secondo, per la selezione di una voce di avvio attraverso la tastiera, trascorso il quale viene scelta automaticamente quella predefinita (che può essere la prima, oppure quella dichiarata con la direttiva '**`default`**'). Lo zero indica di non attendere alcunché, mentre il valore `-1` stabilisce un tempo indefinito. Se non si stabilisce questa direttiva, il tempo predefinito per la pausa è di cinque secondi, pari al valore 50.

- `verbose=`***n***

Permette di stabilire il livello di dettaglio desiderato per le informazioni emesse dall'eseguibile '**`lilo`**'. Si usano valori numerici interi, generalmente da zero a cinque, dove il valore più alto dà informazioni maggiori.

Direttive utilizzabili globalmente e anche nelle sezioni specifiche

Un gruppo di direttive particolari, può essere usato sia in modo particolare, all'interno di sezioni che riguardano le varie voci di avvio, sia in modo globale, prima della dichiarazione di tali sezioni, dove rappresentano l'impostazione predefinita nel caso non vengano utilizzate nuovamente nelle sezioni.

- `append=`***parametri_di_avvio_del_kernel***

Aggiunge la stringa indicata tra i parametri del kernel.

- `initrd=`***file***

Specifica l'uso di un file da caricare all'avvio come disco RAM iniziale.

- `read-only`

Specifica che in fase di avvio il file system deve essere innestato in sola lettura. Ciò è necessario per la verifica e l'eventuale riparazione del file system, quando successivamente il sistema provvede automaticamente a reinnestarlo in lettura e scrittura.

- `read-write`

Specifica che in fase di avvio il file system deve essere innestato in lettura e scrittura.

- `root=`***file***

Indica il file di dispositivo che deve essere innestato come file system principale. Se non si utilizza questa direttiva, si intende implicitamente che si tratti della partizione o del disco in cui si trova già il file del kernel.

- `vga={normal|extended|ask|`***n***`}`

Specifica la modalità video VGA che deve essere impostata all'avvio. La parola chiave '**`normal`**' richiede espressamente la modalità testo normale, pari a 80×25; '**`extended`**' richiede la modalità testo 80×50; '**`ask`**' fa in modo che venga richiesto all'utente in fase di avvio; infine, un valore numerico corrisponde a una scelta equivalente dal menù che si otterrebbe con l'opzione '**`ask`**'.

- `lock`

Questa direttiva abilita la registrazione della riga di comando utilizzata all'avvio, relativa alla propria voce di avvio, allo scopo di riutilizzarla in modo predefinito negli avvii successivi.

- `password=`***parola_d'ordine***

Fa in modo che venga richiesta la parola d'ordine indicata per poter procedere. Naturalmente, occorre tenere presente che il file di configurazione contenente tale informazione, dovrebbe essere protetto in qualche modo, almeno dagli accessi di utenti diversi dall'amministratore.

- `restricted`

Questa direttiva può essere usata solo assieme a '**`password`**' e serve a stabilire che la richiesta di tale parola d'ordine avviene solo nel caso di inserimento di parametri di avvio per il kernel.

- `single-key`

La direttiva '**`single-key`**' consente di avviare un'immagine con la pressione di un solo tasto, senza l'aggiunta di un [*Invio*] finale. Per ottenere questo risultato, si può fare in modo che le varie direttive '**`label`**' definiscano dei nomi composti da un solo carattere, oppure si aggiunge alla direttiva '**`label`**' la direttiva '**`alias`**',

dove però si deve specificare un carattere differente dall'iniziale usata nel nome abbinato a '**label**'. In questo senso, è comune utilizzare delle direttive '**alias**' contenenti solo un numero.

L'avvio attraverso la pressione di un tasto singolo, impedisce l'inserimento di parametri per il kernel. Di conseguenza, per poter selezionare l'avvio, sia con un tasto singolo, sia con un nome, si usano entrambe le direttive '**label**' e '**alias**', con l'accorgimento di non ripetere le iniziali.

Sezioni delle voci di avvio

Le voci selezionabili all'avvio, sono descritte all'interno di sezioni, che hanno la stesso aspetto delle direttive normali. Si tratta precisamente di queste due direttive:

```
image=file_immagine_del_kernel_da_avviare
```

```
other=file_di_dispositivo
```

Nel primo caso si fa riferimento a una sezione relativa a una voce di avvio per un kernel Linux; nel secondo si tratta dell'avvio di un altro settore di avvio, presumibilmente di un sistema operativo diverso da GNU/Linux.

Tutte le direttive successive a una di queste due, fino alla dichiarazione di una sezione successiva eventuale, rappresentano impostazioni particolari. In questo ambito, si possono indicare le direttive già descritte in precedenza, tranne quelle di competenza esclusivamente globale, oltre a quelle che vengono descritte qui in particolare.

- ```
  label=nome
  ```

  Indica il nome attribuito a questa voce di avvio, che può essere selezionato al momento dell'invito.

- ```
  alias=nome
  ```

  Specifica un nome alternativo per la voce a cui si riferisce.

- ```
  loader=file
  ```

  Si usa nell'ambito di una sezione '**other**', per indicare il file contenente il codice necessario per il caricamento di un settore di avvio successivo. In condizioni normali si tratta del file '`/boot/chain.b`', che viene utilizzato in modo predefinito quando non si specifica questa direttiva.

  Se si intende utilizzare più di una sezione '**other**', ognuna riferita a una partizione contenente una copia distinta di uno stesso sistema operativo, o anche di sistemi diversi, può succedere che LILO avvii sempre solo la prima di queste, nonostante il tentativo dell'utente di selezionarne un'altra. Si risolve il problema inserendo in tutte le sezioni '**other**' la direttiva '**loader=/boot/chain.b**'.

- ```
  map-drive=codice_virtuale
      to=codice_reale
  ```

  Queste due direttive, che si usano necessariamente in coppia, specificano lo scambio dei codici indicati, riferiti al BIOS, per ottenere in pratica lo scambio dell'identificazione dei dischi relativi. Ciò si ottiene attraverso il file '`/boot/chain.b`' che installa un programma residente per la gestione di questo scambio, al di sopra del controllo del sistema operativo che si vuole avviare.

  I codici in questione sono tipicamente 80_{16} per il primo disco ATA, 81_{16} per il secondo e così di seguito.

  Si osservi che per ottenere uno scambio completo tra due dischi, occorre usare queste direttive due volte, per entrambi i casi: il primo disco che diventa il secondo e il secondo disco che diventa il primo.

- ```
  table=file_di_dispositivo
  ```

  Specifica, attraverso il file di dispositivo corrispondente, la tabella di partizione relativa al sistema operativo che si intende avviare. Si usa di solito nelle sezioni '**other**', quando non si tratta dell'avvio di GNU/Linux.

Esempi

L'esempio seguente può avviare un sistema GNU/Linux in due modi differenti, attraverso il file '`/boot/vmlinuz`' e '`/boot/vmlinuz.1`', oppure un altro sistema operativo (in questo caso si tratta di MS-Windows).

La presenza di direttive '**alias**', fa sì che si possano selezionare le voci per nome, potendo così aggiungere anche dei parametri per il kernel, oppure attraverso una sola cifra numerica.

Si può osservare che la voce '**linux**', ovvero '**1**', richiede l'inserimento di una parola d'ordine nel caso si vogliano inserire dei parametri di avvio; inoltre, nel caso della voce '**prova**', ovvero '**2**', è impedito l'inserimento di parametri di avvio, attraverso la direttiva '**lock**'.

```
boot=/dev/hda
vga=normal
read-only
prompt
timeout=-1
single-key
message=/boot/message

image=/boot/vmlinuz
  label=linux
  alias=1
  root=/dev/hda4
  initrd=/boot/initrd
  password=segreto
  restricted
image=/boot/vmlinuz.1
  label=prova
  alias=2
  root=/dev/hda4
  initrd=/boot/initrd.1
  lock
other=/dev/hda1
  label=windows
  alias=3
  table=/dev/hda
```

L'estratto seguente, riguarda un gruppo di direttive relative all'avvio di sistemi operativi diversi da GNU/Linux. In particolare, si osserva il fatto che si tenta di avviare OS/2 dal secondo disco fisso PATA, cercando di imbrogliarlo, facendogli credere di essere sul primo. In quel caso particolare si deve usare anche un file speciale nella direttiva '**loader**'.

```
other = /dev/hda2
  label = dos
  table = /dev/hda
other = /dev/hdb2
  label = os2
  loader = /boot/os2_d.b
  map-drive = 0x80
    to = 0x81
  map-drive = 0x81
    to = 0x80
```

L'estratto seguente contiene due sezioni '**other**' per avviare due partizioni distinte contenenti copie indipendenti del sistema MS-Windows. Si osservi in particolare l'uso della direttiva '**loader = /boot/chain.b**' in ogni sezione '**other**'.
```

```
other = /dev/hda1
  label = Win1
  loader = /boot/chain.b
  table = /dev/hda

other = /dev/hda2
  label = Win2
  loader = /boot/chain.b
  table = /dev/hda
```


X

«

X: monitor, adattatore grafico e frequenza dot-clock 263
Autorilevamento con Read-edid 263
Autorilevamento con un servente XFree86 versione 3.* . . . 264
Un po' di teoria 267
Configurazione della sezione «Monitor» di «XF86Config» 272
Rifiniture 275
Altri programmi affini 276
Riferimenti 276

X: monitor, adattatore grafico e frequenza dot-clock

Autorilevamento con Read-edid 263
Autorilevamento con un servente XFree86 versione 3.* 264
Dot-clock 266
Un po' di teoria 267
Ampiezza di banda del monitor 267
Scomposizione e scansione dell'immagine sul monitor 267
Frequenza, durata e lunghezza 269
Definizioni, concetti ed equazioni 269
Multipli di otto e rapporto 3/4 270
Utilizzo della memoria video 271
Impulsi di sincronismo 271
Tradurre i valori in unità dot-clock e in quantità di righe 272
Configurazione della sezione «Monitor» di «XF86Config» 272
Scomposizione delle informazioni 273
Scansione orizzontale 273
Scansione verticale 274
Interlacciamento 274
Adattamento delle configurazioni predefinite 274
Rifiniture 275
Utilizzo di «xvidtune» 276
Altri programmi affini 276
Riferimenti 276

Quando si vuole configurare XFree86 nelle versioni 3.* e qualcosa va storto, oppure non si riesce a ottenere quello che si vuole esattamente attraverso uno dei vari programmi già descritti nel capitolo precedente, può essere necessario mettere mano alle sezioni '**`Monitor`**', '**`Device`**' e '**`Screen`**' del file '`/etc/X11/XF86Config`'. Tra tutte, la sezione '**`Monitor`**' è la più difficile per il principiante, a causa delle direttive '**`Modeline`**' o '**`Mode`**', in cui si devono indicare una serie di numeri più o meno oscuri.

In questo capitolo si mostra in che modo calcolare i valori delle modalità video. Una scelta impropria di questi valori, potrebbe causare problemi, fino ad arrivare al danneggiamento del monitor. Si prega di intervenire con prudenza ed eventualmente anche di leggere *XFree86 Video Timings HOWTO* di Eric S. Raymond.

Autorilevamento con Read-edid

Read-edid [1] è un piccolo sistema di programmi in grado di scandire l'adattatore grafico e il monitor, allo scopo di ottenere le informazioni necessarie a configurare correttamente programmi come XFree86. Si compone precisamente di '**`get-edid`**' e di '**`parse-edid`**'.

Il programma '**`get-edid`**' esegue la scansione dell'adattatore grafico e attraverso di questo anche del monitor. Il risultato della scansione è un file binario emesso attraverso lo standard output, mentre attraverso lo standard error si ottengono altre informazioni diagnostiche. Per esempio, si può ignorare temporaneamente il risultato emesso dallo standard output per osservare tali notizie diagnostiche:

```
# get-edid 1> /dev/null [Invio]
```

Si potrebbe ottenere un risultato simile a quello seguente:

```
get-edid: get-edid version 1.4.1

        Performing real mode VBE call
        Interrupt 0x10 ax=0x4f00 bx=0x0 cx=0x0
        Function supported
```

```
        Call successful

        VBE version 102
        VBE string at 0xc098d "S3 Incorporated. Trio64V+"

VBE/DDC service about to be called
        Report DDC capabilities

        Performing real mode VBE call
        Interrupt 0x10 ax=0x4f15 bx=0x0 cx=0x0
        Function supported
        Call successful

        Monitor and video card combination supports DDC1 transfers
        Monitor and video card combination does not support DDC2 transfers
        0 seconds per 128 byte EDID block transfer
        Screen is blanked during DDC transfer

Reading next EDID block

VBE/DDC service about to be called
        Read EDID

        Performing real mode VBE call
        Interrupt 0x10 ax=0x4f15 bx=0x1 cx=0x0
        Function supported
        Call successful
```

In questo caso è stato individuato un vecchio adattatore grafico «Trio64V+».

Per poter leggere il risultato emesso attraverso lo standard error, si usa '**`parse-edid`**':

```
# get-edid 2> /dev/null | parse-edid [Invio]
```

Ecco quello che si potrebbe ottenere:

```
        # EDID version 1 revision 0
Section "Monitor"
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
        Identifier "PHL:0012"
        VendorName "PHL"
        ModelName "PHL:0012"
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
        # DPMS capabilities: Active off:yes  Suspend:yes  Standby:yes

        Mode    "640x400"       # vfreq 70.072Hz, hfreq 31.462kHz
                DotClock        25.170000
                HTimings        640 656 752 800
                VTimings        400 412 414 449
                Flags   "+HSync" "-VSync"
        EndMode
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
        # Block type: 2:0 3:0
EndSection
```

In pratica, con questo risultato si può compilare la sezione '**`Monitor`**' del file di configurazione di XFree86.

Autorilevamento con un servente XFree86 versione 3.*

«

Quando non si conoscono tutte le caratteristiche del proprio adattatore grafico, è possibile utilizzare un servente X con l'opzione '**`-probeonly`**' per vedere cosa questo riesce a determinare da solo. Alcuni parametri sono sensibili al carico del sistema, per cui, questo tipo di prova deve essere fatto quando non si effettuano altre attività.

Qui si sta facendo riferimento all'uso di un servente XFree86 versione 3.*. Pertanto, il file di configurazione di partenza che viene proposto, rispecchia la sintassi relativa a quelle versioni e non può funzionare per una versione 4.*.

È il caso di utilizzare un servente più o meno generico, per esempio quello per gli adattatori SVGA ('**`XF86_SVGA`**'), che deve essere stato installato. Per stimolare l'autorilevamento, è necessario che le voci corrispondenti non siano presenti nel file di configurazione '`/etc/X11/XF86Config`' (o siano commentate). Un file come quello seguente, dove le sezioni '**`Monitor`**', '**`Device`**' e '**`Screen`**' sono quasi

vuote, dovrebbe andare bene per cominciare lo studio del proprio adattatore grafico:

```
Section "Files"
    RgbPath     "/etc/X11/rgb"
    FontPath    "/usr/lib/X11/fonts/misc/"
    FontPath    "/usr/lib/X11/fonts/Type1/"
    FontPath    "/usr/lib/X11/fonts/Speedo/"
    FontPath    "/usr/lib/X11/fonts/75dpi/"
    FontPath    "/usr/lib/X11/fonts/100dpi/"
EndSection

Section "ServerFlags"
    # DontZap
    # DontZoom
EndSection

Section "Keyboard"
    Protocol    "Standard"
    AutoRepeat  500 5
    Xkbkeycodes "xfree86"
    XkbTypes    "default"
    XkbCompat   "default"
    XkbSymbols  "en_US(pc102)+it"
    XkbGeometry "pc"
EndSection

Section "Pointer"
    Protocol    "microsoft"
    Device      "/dev/mouse"
    Emulate3Buttons
    Emulate3Timeout    50
EndSection

Section "Monitor"
    Identifier  "Monitor generico"
EndSection

Section "Device"
    Identifier  "SuperVGA"
EndSection

Section "Screen"
    Driver      "svga"
    Device      "SuperVGA"
    Monitor     "Monitor generico"
    Subsection "Display"
        Modes       "640x400" "640x480" "640x480.28" "800x600"
    EndSubsection
EndSection
```

Si avvia quindi '**`X`**', come utente '**`root`**', con l'opzione '**`-probeonly`**', salvando lo standard output e lo standard error in un file ('**`X`**' è un collegamento simbolico al file binario del servente grafico prescelto).

Purtroppo, è necessario tenere in considerazione che questo tipo di prove può modificare l'aspetto dei caratteri sullo schermo, o bloccarlo del tutto. Per cui, se non si hanno alternative, si rischia di dover riavviare il sistema.

```
# X -probeonly > /tmp/x.tmp 2>&1 [Invio]
```

Se tutto è andato bene, il file '`/tmp/x.tmp`' generato dal comando, dovrebbe contenere un risultato simile a quello seguente, che viene sezionato per descriverlo in dettaglio.

```
XFree86 Version 3.3.2 / X Window System
(protocol Version 11, revision 0, vendor release 6300)
Release Date: March 2 1998
        If the server is older than 6-12 months, or if your card is newer
        than the above date, look for a newer version before reporting
        problems.  (see http://www.XFree86.Org/FAQ)
Operating System: Linux 2.0.34 i686 [ELF]
```

La parte iniziale presenta la versione del servente e del sistema operativo utilizzato.

```
Configured drivers:
  SVGA: server for SVGA graphics adaptors (Patchlevel 0):
      NV1, STG2000, RIVA128, ET4000, ET4000W32, ET4000W32i,
      ET4000W32i_rev_b, ET4000W32i_rev_c, ET4000W32p, ET4000W32p_rev_a,
…
      s3_svga, ct65520, ct65525, ct65530, ct65535, ct65540, ct65545,
      ct65546, ct65548, ct65550, ct65554, ct65555, ct68554, ct64200,
      ct64300, generic
```

Segue quindi l'indicazione del tipo di servente grafico avviato (SVGA) e l'elenco di tutti i nomi degli adattatori grafici gestibili con questo.

```
(using VT number 7)
```

Il servente grafico utilizzerebbe (se avviato normalmente) il posto della console virtuale numero sette.

```
XF86Config: /usr/lib/X11/XF86Config
```

È stata letta la configurazione del file '`/usr/lib/X11/XF86Config`' (in questo caso si tratta di un collegamento simbolico a '`/etc/X11/XF86Config`').

Dopo questo punto segue un elenco di informazioni, in parte definite all'interno del file di configurazione e in parte determinate in modo automatico.

```
(**) stands for supplied, (--) stands for probed/default values
```

Le informazioni fornite attraverso il file di configurazione sono prefissate dal simbolo '**`(**)`**', mentre quelle predefinite o determinate dall'interrogazione dell'adattatore grafico, sono prefissate dal simbolo '**`(--)`**'.

```
(**) XKB: keycodes: "xfree86"
(**) XKB: types: "default"
(**) XKB: compat: "default"
(**) XKB: symbols: "en_US(pc102)+it"
(**) XKB: geometry: "pc"
(**) Mouse: type: microsoft, device: /dev/mouse, baudrate: 1200
(**) Mouse: buttons: 3, 3 button emulation (timeout: 50ms)
(**) SVGA: Graphics device ID: "SuperVGA"
(**) SVGA: Monitor ID: "Monitor generico"
(**) FontPath set to "/usr/lib/X11/fonts/misc/,…
```

Dato l'esempio proposto, le informazioni sulla tastiera, il mouse e i percorsi dei tipi di carattere, sono prelevati dal file di configurazione. In particolare, si osserva che da quel file, sono state prese in considerazione la sezione '**`Device`**' denominata '**`SuperVGA`**' e la sezione '**`Monitor`**' denominata '**`Monitor generico`**'.

```
(--) SVGA: PCI: S3 ViRGE/DX or /GX rev 1, Memory @ 0xe0000000
(--) SVGA: S3V: ViRGE/DXGX rev 1, Linear FB @ 0xe0000000
(--) SVGA: Detected S3 ViRGE/DXGX
(--) SVGA: using driver for chipset "s3_virge"
```

L'adattatore grafico è una scheda S3 ViRGE/DXGX, per la quale verrebbe utilizzato il driver '**`s3_virge`**'. Tuttavia, data la circostanza, converrebbe utilizzare un servente grafico differente per questo adattatore; precisamente '**`XF86_S3V`**'.

```
(--) SVGA: videoram:  4096k
(--) SVGA: Ramdac speed: 170 MHz
(--) SVGA: Detected current MCLK value of 42.955 MHz
(--) SVGA: chipset:  s3_virge
(--) SVGA: videoram: 4096k
(**) SVGA: Using 8 bpp, Depth 8, Color weight: 666
(--) SVGA: Maximum allowed dot-clock: 170.000 MHz
```

Seguono altre informazioni molto importanti, come la quantità di memoria video e la frequenza massima di dot-clock. Si osservi in particolare la profondità di colori indicata: 8 bit/pixel (8 bit per punto). L'informazione è preceduta dal simbolo '**`(**)`**' perché il tipo di servente grafico permette la gestione di un massimo di 8 bit/pixel (256 colori), per cui è questo il valore fissato, benché l'adattatore grafico permetta ben altri livelli di profondità.

Dot-clock

Una delle informazioni più delicate dell'adattatore grafico è la frequenza del cosiddetto ***dot-clock***. Il significato di questo parametro viene descritto più avanti, tuttavia è bene sapere subito che si può manifestare in modi differenti.

Nell'esempio mostrato, appare l'indicazione di un livello massimo.

```
(--) SVGA: Maximum allowed dot-clock: 170.000 MHz
```

In altre situazioni, può essere fornita una o più righe con un elenco di valori di dot-clock, come nell'esempio seguente:

```
(--) xxx: clocks:  25.0  28.0  40.0   0.0  50.0  77.0  36.0  45.0
(--) xxx: clocks: 130.0 120.0  80.0  31.0 110.0  65.0  75.0  94.0
```


In questo secondo caso, è necessario indicare la direttiva '**`Clocks`**' nella sezione '**`Device`**' del file '`/etc/X11/XF86Config`', come nell'esempio seguente:

```
Section "Device"
    …
    Clocks    25.0  28.0  40.0   0.0  50.0  77.0  36.0  45.0
    Clocks   130.0 120.0  80.0  31.0 110.0  65.0  75.0  94.0
EndSection
```

Quando invece la frequenza di dot-clock viene indicata solo come valore massimo (come nel caso dell'adattatore S3 ViRGE), non serve indicare alcuna direttiva '**`Clocks`**'.

Un po' di teoria

Alla base della costruzione dell'immagine da parte dell'adattatore grafico, sta la frequenza di dot-clock, ovvero la frequenza a cui ogni punto che la compone viene emesso. Questa è espressa in megahertz (MHz) e a volte deve essere selezionata da un elenco (quando si deve utilizzare la direttiva '**`Clocks`**'), altre volte può essere programmata liberamente, purché non venga superato il limite massimo.

In linea di massima, l'adattatore grafico VGA elementare tradizionale, ha una frequenza di dot-clock di 25175 MHz.

Chi lavora con l'informatica potrebbe essere portato a confondersi. In questo caso, megahertz significa esattamente milioni di hertz. Per cui, 25175 MHz sono esattamente pari a 25175000 Hz. Così, kilohertz rappresenta migliaia di hertz, per cui, per esempio, 31,5 kHz corrispondono a 31500 Hz.

A parità di condizioni, al crescere della risoluzione deve crescere la frequenza di dot-clock. Leggendo il contenuto standard di una vecchia versione 3.* del file '`/etc/X11/XF86Config`', si conoscono i valori minimi delle frequenze di dot-clock per le risoluzioni più comuni. Qui vengono riportate nella tabella u50.15.

Tabella u50.15. Frequenze minime di dot-clock in base alla risoluzione.

Risoluzione	Frequenza di dot-clock minima
640×480	25,175 MHz
800×600	36 MHz
1024×768 interlacciato	44,9 MHz
1024×768	65 MHz
1152×864 interlacciato	65 MHz
1152×864	92 MHz
1280×1024 interlacciato	80 MHz
1280×1024	110 MHz
1600×1200	162 MHz
1800×1440	230 MHz

Ampiezza di banda del monitor

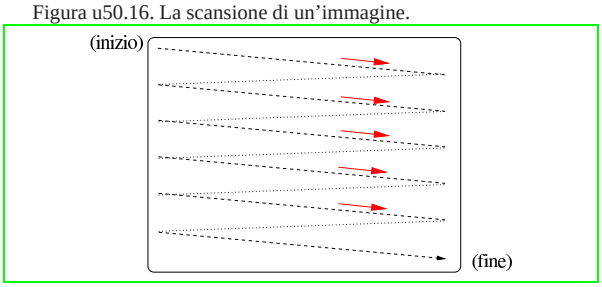
L'ampiezza di banda del monitor, o *bandwidth*, rappresenta la frequenza massima del segnale video che il monitor è in grado di gestire. Frequenze superiori vengono semplicemente filtrate, diventando particolari visivi non più percettibili.

In linea di principio, la frequenza di dot-clock utilizzata nell'adattatore grafico dovrebbe essere inferiore o uguale al valore massimo della frequenza del segnale video gestibile con il monitor, cioè al valore dell'ampiezza di banda.

Scomposizione e scansione dell'immagine sul monitor

L'immagine che appare sullo schermo di un monitor può essere descritta, in modo semplificato, come l'insieme di una serie di righe, composte a loro volta da punti. La prima forma di rappresentazione di un'immagine di origine elettronica è stata quella del tubo a raggi catodici e da questo tipo di tecnologia derivano le soluzioni adottate per la sua composizione.

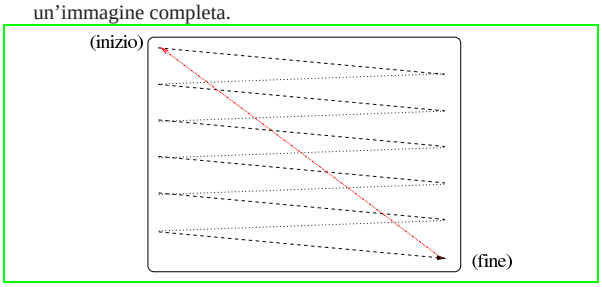
Figura u50.16. La scansione di un'immagine.



Le righe di un'immagine video vengono disegnate da un «pennello» ideale, che inizia la sua scansione in una posizione dello schermo in alto a sinistra, muovendosi verso destra e ricominciando sempre dal lato sinistro della riga successiva. Giunto alla fine dello schermo, riprende dalla posizione superiore sinistra.

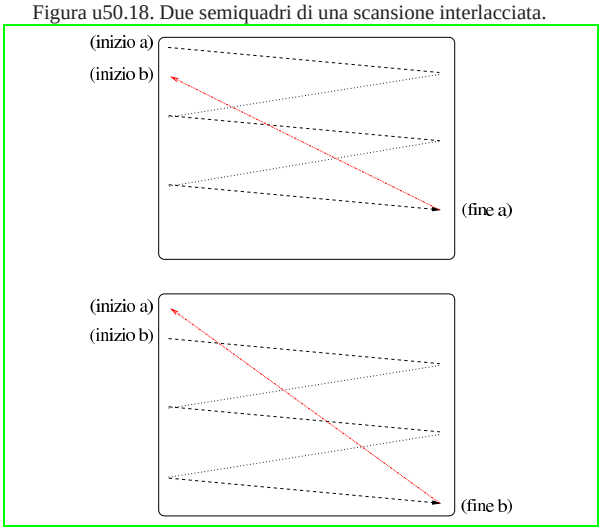
Il pennello di scansione, una volta che ha terminato una riga, prima di poter riprendere con la riga successiva, deve avere il tempo necessario per posizionarsi all'inizio di questa. Nello stesso modo, quando il pennello di scansione giunge alla fine dell'ultima riga, deve avere il tempo di ritornare all'inizio dello schermo, cioè nella posizione estrema in alto a sinistra.

Figura u50.17. Il ritorno all'inizio dopo la scansione di un'immagine completa.



Un'immagine completa è un ***quadro***, o *frame*, ma un quadro potrebbe essere ottenuto con un'unica scansione, dall'inizio alla fine dello schermo, oppure dalla somma di due ***semiquadri***. In questo ultimo caso si usa la tecnica dell'interlacciamento, in cui le righe dei due semiquadri si affiancano senza accavallarsi. La figura u50.18 mostra il caso di un quadro composto da un numero di righe pari.

Figura u50.18. Due semiquadri di una scansione interlacciata.

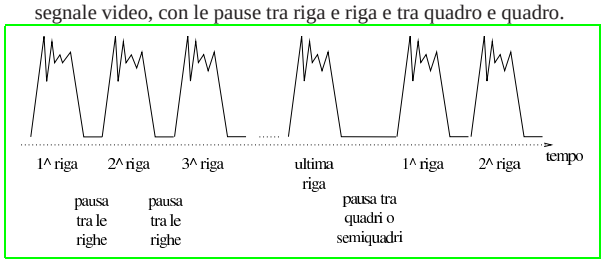


L'interlacciamento è nato come un metodo per ridurre lo sfarfallio dell'immagine nel sistema televisivo tradizionale. Per esempio, in Europa i quadri si susseguono a una frequenza di 25 Hz, un valore troppo basso perché l'occhio umano non si accorga dello sfarfallio. Così, attraverso l'interlacciamento, le immagini trasmesse vengono scomposte in due parti, visualizzate in sequenza a una frequenza di 50 Hz, considerata accettabile per quel tipo di utilizzo, anche se questo può comunque provocare strani effetti alla percezione dei particolari.

In generale, a parità di frequenza di quadro, è preferibile un'immagine interlacciata per ridurre l'effetto dello sfarfallio.

Da quanto affermato si può intendere che l'immagine video sia prodotta come una sequenza lineare di punti e di pause, necessarie al ritorno all'inizio di una riga successiva, di un quadro o di un semiquadro successivo.

Figura u50.19. Rappresentazione schematica dello scorrere del segnale video, con le pause tra riga e riga e tra quadro e quadro.



Il monitor su cui si visualizza il segnale video, deve avere un modo per sapere quando inizia un quadro e quando inizia ogni riga. Le pause necessarie al ritorno del pennello di scansione, vengono usate per sincronizzare la scansione stessa.

Frequenza, durata e lunghezza

La frequenza di dot-clock è una sorta di orologio che scandisce il tempo del segnale video. Un ciclo di questa frequenza rappresenta un punto dell'immagine. Questa frequenza si esprime in megahertz, per cui, una frequenza di dot-clock di 25,175 indica che in un secondo possono essere visualizzati 25 175 000 punti (si deve tenere presente che si tratta di valori teorici).

Seguendo questo ragionamento, le «misure» dell'immagine potrebbero essere valutate in quantità di dot-clock.

In tutto si utilizzano tre tipi di unità di misura per ciò che riguarda la composizione delle immagini: frequenze, riferite ai cicli di scansione delle righe e dei quadri; durate, riferite alle pause tra le righe e tra i quadri; lunghezze, pari alla traduzione di questi valori in unità di dot-clock.

Ricapitolando quanto già esposto nella sezione precedente, l'immagine video è composta da quadri che a loro volta si scompongono in righe. Le righe vengono scandite a una certa frequenza, definita come ***frequenza orizzontale***, e così anche i quadri, ***frequenza di quadro***. Queste frequenze possono essere tradotte in «lunghezze», riferite a unità di dot-clock. Per esempio, la frequenza orizzontale di 31,5 kHz (31 500 Hz), misurata con un dot-clock di 25,175 MHz, si traduce in una lunghezza di riga pari a 799,2 punti (25 175 000 / 31 500 = 799,2).

Quando si valutano grandezze riferite alla scansione verticale dell'immagine, per esempio la frequenza di quadro, si utilizzano lunghezze riferite al numero di righe. Continuando l'esempio precedente, se si aggiunge che la frequenza verticale è di 60 Hz, si determina che un quadro è composto da circa 419 583 dot-clock, pari a circa 525 righe.

Come già affermato, anche lo scorrere del tempo può essere valutato in unità dot-clock. Per esempio, un intervallo di tempo di 3,8 μs (microsecondi, ovvero milionesimi di secondo) è lungo 95,6 dot-clock (25 175 000 * 0,000 003 8 = 95,6).

Definizioni, concetti ed equazioni

La documentazione di XFree86 utilizza alcune definizioni che conviene elencare e chiarire. Le sigle indicate fanno volutamente riferimento a quelle utilizzate nel *XFree86 Video Timings HOWTO*.

- ***Frequenza di sincronizzazione orizzontale***, *Horizontal sync frequency*, HSF

La frequenza di scansione orizzontale delle righe sullo schermo. Generalmente si tratta di una grandezza espressa in kilohertz.

- ***Frequenza di sincronizzazione verticale***, *Vertical sync frequency*, VSF

  La frequenza di scansione verticale dei semiquadri, o dei quadri, sullo schermo. Quando l'immagine viene composta attraverso semiquadri interlacciati, si tratta della frequenza di scansione dei semiquadri stessi, altrimenti si tratta della stessa frequenza di scansione dei quadri interi.

- ***Frequenza di quadro***, ***frequenza di frame***, *Vertical refresh rate*, RR

  La frequenza di scansione verticale dei quadri interi (*frame*).

- ***Frequenza di dot-clock***, *Dot-clock frequency*, DCF

  La frequenza di dot-clock, espressa generalmente in megahertz.

- ***Ampiezza di banda video***, *Video bandwidth*, VB

  La frequenza massima per il segnale video accettato dal monitor. Questo valore dovrebbe essere maggiore o uguale a quello del dot-clock, ma anche valori inferiori a questo permettono ugualmente di vedere qualcosa. In generale, il livello minimo dell'ampiezza di banda deve essere almeno superiore alla metà della frequenza di dot-clock.

- ***Ampiezza orizzontale***, *Horizontal frame length*, HFL

  Si tratta della lunghezza totale di una riga, espressa in dot-clock. Questa dimensione deve includere la parte visibile e la pausa prima dell'inizio della riga successiva.

- ***Ampiezza verticale***, *Vertical frame length*, VFL

  Si tratta dell'altezza totale di un quadro intero, espressa in righe. Questa dimensione deve includere la parte visibile e la pausa prima dell'inizio del quadro successivo.

- ***Risoluzione orizzontale***, *Horizontal resolution*, HR

  La risoluzione orizzontale, espressa in punti o dot-clock, della parte visibile dell'immagine. Per definizione, si tratta di un valore inferiore dell'ampiezza orizzontale (HFL).

- ***Risoluzione verticale***, *Vertical resolution*, VR

  La risoluzione verticale, espressa in righe, della parte visibile dell'immagine. Per definizione, si tratta di un valore inferiore dell'ampiezza verticale (VFL).

Alcune equazioni elementari possono aiutare a collegare i vari pezzi del mosaico.

- HSF = DCF / HFL

  La frequenza di scansione orizzontale equivale alla frequenza di dot-clock divisa per la lunghezza completa della riga (espressa in dot-clock).

- RR = DCF / (HFL * VFL)

  La frequenza di scansione di un quadro intero equivale alla frequenza di dot-clock divisa per il prodotto della lunghezza completa della riga e il numero complessivo delle righe.

  La frequenza di sincronizzazione verticale è pari al doppio di RR quando si utilizzano semiquadri interlacciati, altrimenti corrisponde al valore di RR.

- DCF = RR * HFL * VFL

  Derivata dalla precedente. La frequenza di dot-clock si ottiene con il prodotto della frequenza di scansione di un quadro intero, la lunghezza completa della riga e il numero complessivo delle righe.

Multipli di otto e rapporto 3/4

Una particolarità comune dei valori che riguardano la risoluzione di un'immagine, è l'essere un multiplo di otto. Se si osserva, valori come 640×480, 800×600, 1024×768,... sono numeri divisibili per otto, senza lasciare alcun resto.

Un gran numero di adattatori grafici accetta determinati tipi di valori solo se sono multipli di otto. Per questo, in generale, per tutte le «lunghezze» orizzontali, quindi ciò che si esprime in punti o in dot-clock e riguarda la riga, si deve avere l'accortezza di usare multipli di otto. Questo particolare viene chiarito meglio in seguito.

Data la tradizione televisiva, il formato più comune della visualizzazione su monitor è 3/4, cioè la risoluzione verticale (il numero delle righe visibili) è il 75 % rispetto alla risoluzione orizzontale (il numero di punti visibili per riga). Questa regola non è obbligatoria. L'unico vincolo sono i multipli di otto per le grandezze che riguardano la scansione orizzontale.

Utilizzo della memoria video

L'immagine che appare sullo schermo di un monitor viene generata all'interno di una matrice contenuta in una memoria, che poi viene scandita nel modo che è stato spiegato. All'interno di questa memoria si deve conservare solo la parte di immagine visibile effettivamente, escludendo le pause inserite per facilitare il compito del pennello di scansione del monitor.

La memoria disponibile pone un limite alla risoluzione massima e alla ***profondità*** dell'immagine. A seconda del numero di colori o di sfumature che si vogliono rappresentare, deve essere impiegato un numero più o meno grande di bit per ogni punto dell'immagine. Se ***n*** è il numero di bit messo a disposizione per ogni punto, il numero di colori o sfumature disponibile è di 2^n. Nello stesso modo, conoscendo la memoria disponibile e la risoluzione che si vuole ottenere, si determina quanti siano i colori ottenibili per ogni punto.

Per esempio, se si dispone di 1 Mibyte, pari a 1048576 byte, cioè 8388608 bit, volendo ottenere una risoluzione (visibile) di 800×600 punti, si ottiene che per ogni punto sarebbero disponibili 17 bit (8388608 / (800 * 600)).

Tuttavia, di solito, il numero di bit che può essere utilizzato per definire la profondità di un'immagine è limitato a valori ben precisi: 2 bit (bianco/nero), 4 bit (16 colori), 8 bit (256 colori), 16 bit (64 Kicolori), 24 bit (16 Micolori), 32 bit (4 Micolori),...

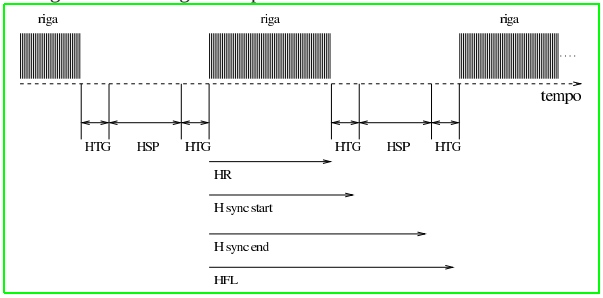
Si osservi che in alcune circostanze, vale anche per la profondità di colori la regola del multiplo di otto; per esempio, di solito si ha a che fare con profondità da 8 bit, 16 bit, 24 bit e 32 bit.

Impulsi di sincronismo

Fino a questo momento sono state descritte le immagini video come qualcosa formato da righe visibili, collegate tra loro da delle pause, a formare dei quadri (o dei semiquadri), i quali si collegano tra loro con delle pause più grandi. Quando si è accennato ai concetti di ampiezza orizzontale e verticale, si è sottolineato il fatto che queste grandezze devono includere anche le pause relative.

Ma le pause da sole non bastano. Al loro interno si inseriscono degli impulsi di sincronismo, senza i quali queste non sarebbero riconosciute dal monitor.

Figura u50.20. Righe e impulsi di sincronismo.



L'impulso di sincronismo orizzontale ha una durata che può variare da monitor a monitor, ma in ogni caso si esprime in un'unità di tempo che resta costante al variare della frequenza dot-clock. Generalmente sono accettabili valori compresi tra 3,5 μs e 4,0 μs (microsecondi). La figura u50.20 mostra schematicamente gli elementi che compongono una riga completa: la parte visibile, definita come risoluzione orizzontale (HR), un tempo di guardia precedente all'impulso di sincronismo (HTG), l'impulso di sincronismo (HSP), un tempo di guardia finale. Quindi ricomincia un'altra riga.

Il tempo di guardia iniziale e finale è importante come l'impulso di sincronismo, tuttavia viene definito normalmente in modo approssimativo, salvo aggiustamenti successivi. In generale, un tempo di guardia medio di 30 dot-clock dovrebbe andare bene. È importante osservare subito che di solito il tempo di guardia iniziale e finale non sono simmetrici.

In maniera analoga funziona il sincronismo verticale. Si ha un tempo di guardia iniziale (VTG, *Vertical time guard*), un impulso di sincronismo verticale (VSP) e un tempo di guardia finale. L'impulso di sincronismo dovrebbe oscillare tra i 50 μs e i 300 μs (microsecondi).

Tradurre i valori in unità dot-clock e in quantità di righe

La definizione dei vari elementi che compongono l'immagine deve essere fatta attraverso due unità di misura uniformi: dot-clock per ciò che riguarda la scansione orizzontale e righe per la scansione verticale.

Si è accennato al fatto che il tempo di guardia orizzontale può aggirarsi attorno a un valore di 30 dot-clock, senza bisogno di fare altri calcoli, mentre il problema si pone per trasformare il tempo dell'impulso di sincronismo in dot-clock. Basta moltiplicare la frequenza di dot-clock per il tempo. La frequenza è espressa in hertz e il tempo in secondi.

Lunghezza in dot-clock = DCF * tempo

Per riprendere un esempio già fatto, se si utilizza una frequenza di dot-clock di 25,175 MHz e si vuole misurare un intervallo di 3,8 μs, si ottiene una lunghezza di 95,6 dot-clock (25 175 000 * 0,000 003 8).

Il vero problema, quando si fa riferimento a grandezze orizzontali, è il fatto che queste devono essere espresse in multipli di otto. Molte approssimazioni nei calcoli relativi, che per il momento non sono ancora state mostrate, derivano da questa esigenza.

Il tempo di guardia verticale, a seconda del tipo di monitor utilizzato, potrebbe essere assente del tutto, oppure potrebbe essere richiesto un massimo di tre righe. Eventualmente, un tempo di guardia maggiore del necessario, non può essere dannoso.

Il calcolo della lunghezza dell'impulso di sincronismo verticale, in termini di righe, è un po' più complesso. Uno dei modi possibili è quello di definire prima la lunghezza in dot-clock e quindi di convertirla in righe, dividendo questo valore per la lunghezza complessiva della riga.

Lunghezza VSP = (DCF * tempo) / HFL

Riprendendo l'esempio precedente, aggiungendo che una riga ha la lunghezza complessiva di 800 dot-clock, volendo calcolare un impulso di sincronismo verticale di 64 μs circa, si ottengono due righe ((25 175 000 * 0,000 064) / 800).

Configurazione della sezione «Monitor» di «XF86Config»

Quanto descritto fino a questo momento serve per chiarire il significato delle direttive contenute nella sezione '**`Monitor`**' del file di configurazione di XFree86: '`/etc/X11/XF86Config`'. Viene proposto un esempio:

```
Section "Monitor"
    Identifier  "Monitor generico"
    HorizSync   31.5, 35.15
    VertRefresh 50-70

    # 640x400 @ 70 Hz, 31.5 kHz hsync
    Modeline "640x400"     25.175 640  664  760  800   400  409  411  450

    # 640x480 @ 60 Hz, 31.5 kHz hsync
    Modeline "640x480"     25.175 640  664  760  800   480  491  493  525

    # 800x600 @ 56 Hz, 35.15 kHz hsync
    Modeline "800x600"     36     800  824  896 1024   600  601  603  625
EndSection
```

Si deve osservare che ogni direttiva '**`Modeline`**', o la sua equivalente '**`Mode`**', contiene tutte le informazioni necessarie sul funzionamento del monitor in corrispondenza a quella particolare modalità. Questo significa che le direttive '**`HorizSync`**' e '**`VertRefresh`**' sono solo un'informazione aggiuntiva che serve a permettere un controllo incrociato. Per essere più precisi, il file '`/etc/X11/XF86Config`' potrebbe contenere informazioni su molte modalità di visualizzazione, che vengono selezionate in base alle limitazioni poste dalle direttive '**`HorizSync`**' e '**`VertRefresh`**'.

Scomposizione delle informazioni

La direttiva '**`Modeline`**' contiene una serie di notizie che è necessario distinguere per poterne conoscere il significato.

```
Modeline nome freq_dot_clock informazioni_scansione_orizz informazioni_scansione_ve
```

In particolare, le informazioni sulla scansione orizzontale e quelle sulla scansione verticale sono una coppia di quattro numeri distinti (otto in tutto).

```
#          nome          dot          scansione           scansione
#        modalità      clock        orizzontale          verticale
#                                 ------------------   ------------------
Modeline "640x480"     25.175  640  664  760  800   480  491  493  525
```

Le opzioni sono costituite da parole chiave che possono apparire in coda, in presenza di occasioni particolari, secondo quanto descritto nella documentazione del servente grafico che si utilizza.

È bene ripetere che la direttiva '**`Modeline`**' potrebbe essere sostituita con ***Mode***, una specie di sottosezione molto più leggibile.

```
Mode nome
    DotClock     frequenza_dot_clock
    HTimings     informazioni_scansione_orizzontale
    VTimings     informazioni_scansione_verticale
    [Flags       opzioni…]
EndMode
```

Segue l'esempio già mostrato sopra.

```
Mode "640x480"
    DotClock    25.175
    HTimings    640  664  760  800
    VTimings    480  491  493  525
EndMode
```

Scansione orizzontale

I quattro valori indicati nella direttiva '**`HTimings`**', o quelli che appaiono subito dopo la frequenza di dot-clock nella direttiva '**`Modeline`**', rappresentano i tempi della scansione orizzontale, espressi in unità di dot-clock.

```
risoluzione_orizzontale  inizio_sinc  fine_sinc  ampiezza_orizzontale
```

In pratica, seguendo l'esempio già mostrato, «640 664 760 800» indica che: la risoluzione orizzontale è di 640 punti, o dot-clock, l'impulso di sincronismo orizzontale inizia in corrispondenza del 664-esimo dot-clock e termina con il 760-esimo dot-clock, infine la lunghezza complessiva della riga è di 800 punti.

Con qualche conto si scopre che la frequenza orizzontale necessaria per la scansione con questa modalità è di 31,5 kHz

(25175000 / 800) e che la durata dell'impulso di sincronismo è di 3,8 μs ((760 − 664) / 25175000).

La cosa più importante da osservare è che tutti i valori sono divisibili per otto.

Scansione verticale

I quattro valori indicati nella direttiva '**VTimings**', o gli ultimi quattro valori della direttiva '**Modeline**', rappresentano i tempi della scansione verticale, espressi in quantità di righe.

risoluzione_verticale inizio_sinc fine_sinc ampiezza_verticale

In pratica, seguendo l'esempio già mostrato, «480 491 493 525» indica che: la risoluzione verticale è di 480 righe, l'impulso di sincronismo verticale inizia in corrispondenza della 491-esima riga (ideale) e termina con la 493-esima, infine l'altezza complessiva del quadro è di 525 righe.

Con qualche conto si scopre che la frequenza verticale (del quadro intero) necessaria per la scansione con questa modalità è di 70 Hz (25175000 / (800 * 525)) e che la durata dell'impulso di sincronismo è di 64 μs ((493 − 491) * 800 / 25175000).

La frequenza dei semiquadri è doppia, quando si utilizza una modalità interlacciata. Questo va tenuto in considerazione, perché è la frequenza dei semiquadri quella che viene presa in considerazione nella direttiva '**VertRefresh**'.

Interlacciamento

La predisposizione di una modalità interlacciata richiede solo due particolarità: che il numero complessivo delle righe (VFL) sia in numero dispari e che si aggiunga alla fine l'opzione '**Interlace**'.

```
# 1024x768 @ 87 Hz interlaced, 35.5 kHz hsync
Modeline "1024x768"  44.9 1024 1048 1208 1264   768  776  784  817 Interlace

# 1152x864 @ 89 Hz interlaced, 44 kHz hsync
ModeLine "1152x864"   65  1152 1168 1384 1480   864  865  875  985 Interlace

# 1280x1024 @ 87 Hz interlaced, 51 kHz hsync
Modeline "1280x1024"  80  1280 1296 1512 1568  1024 1025 1037 1165 Interlace
```

I valori riferiti alla scansione verticale si riferiscono sempre al quadro intero, per cui, la frequenza di sincronizzazione verticale risulta doppia rispetto alla frequenza di quadro (*refresh rate* o *frame rate*).

A questo si può aggiungere che la durata dell'impulso di sincronismo verticale dovrebbe essere doppia (o quasi) rispetto a quella necessaria in caso di scansione normale (non interlacciata).

Adattamento delle configurazioni predefinite

Il file di configurazione di XFree86, '`/etc/X11/XF86Config`', offre molti esempi validi di configurazione del monitor, ma non tutti i casi possibili e immaginabili. Uno degli elementi che può creare disturbo è proprio la frequenza di dot-clock.

È già stato spiegato che il servente grafico, usato con l'opzione '**-probeonly**', può dare tante informazioni utili sull'adattatore grafico utilizzato. Tra le altre cose, dovrebbe essere in grado di informare sulle frequenze di dot-clock disponibili. Quello che si vede dall'esempio è l'informazione sul dot-clock di un elaboratore portatile Zenith (Z*Star 433VL), ottenuto da un servente XFree86 versione 3.*.

```
(--) VGA16: clocks: 28.32 28.32 28.32 28.32
```

Potrebbe nascere un problema se si tratta di frequenze fisse che non corrispondono ad alcuna modalità predefinita del file di configurazione; proprio come nel caso dell'esempio.

Intuitivamente, si può cercare di adattare una modalità che abbia una frequenza di dot-clock abbastanza vicina. Osservando il file di configurazione predefinito si possono trovare queste due modalità.


```
# 640x480 @ 60 Hz, 31.5 kHz hsync
Modeline "640x480"     25.175 640  664  760  800   480  491  493  525

# 640x480 @ 72 Hz, 36.5 kHz hsync
Modeline "640x480"     31.5   640  680  720  864   480  488  491  521
```

Anche se non si conosce nulla delle caratteristiche del monitor (in questo caso è quello del portatile, un LCD) si può azzardare l'idea che delle frequenze orizzontali e verticali comprese tra i valori di questi esempi, non dovrebbero creare problemi (la frequenza orizzontale di 31,5 kHz è quella più bassa in assoluto rispetto a tutte le modalità predefinite). Si procede per tentativi.

Evidentemente, dagli esempi proposti, ci si accontenta di una risoluzione di 640×480 punti, quindi questi valori sono noti. Inoltre, si può decidere di mantenere le stesse frequenze di sincronizzazione verticale e orizzontale dell'esempio già visto che utilizzava una frequenza di dot-clock leggermente più bassa. Così facendo, la pausa tra una riga e l'altra dovrebbe aumentare, come probabilmente anche la pausa tra un quadro e l'altro.

```
# 640x480 @ 60 Hz, 31.5 kHz hsync
Modeline "640x480"     28.32  640  ???  ???  ???   480  ???  ???  ???
```

Conoscendo la frequenza di scansione orizzontale, si calcola la dimensione complessiva della riga: 28320000 / 31500 = 899, ma questo numero deve essere divisibile per otto, così si sceglie il valore 896. Nello stesso modo si calcola il numero di righe complessivo che compone un quadro: (28320000 / 896) / 60 = 526,78, ma si sceglie di approssimare per difetto (al massimo, la frequenza verticale diviene leggermente più alta di 60 Hz).

```
# 640x480 @ 60 Hz, 31.5 kHz hsync
Modeline "640x480"     28.32  640  ???  ???  896   480  ???  ???  526
```

Adesso è la volta di determinare la durata dell'impulso di sincronismo orizzontale. Dovrebbe essere di circa 4 μs: 28320000 * 0,000004 = 113 dot-clock. Il problema adesso è quello di trovare qualcosa di soddisfacente che sia divisibile per otto.

((896 − 640) − 113) / 2 = 71,5

640 + 71 = 711; il valore più vicino che sia divisibile per otto è 712.

712 + 113 = 825; il valore più vicino che sia divisibile per otto è 824.

896 − 824 = 72, che rende il tempo di guardia perfettamente simmetrico (è stato solo un caso).

```
# 640x480 @ 60 Hz, 31.5 kHz hsync
Modeline "640x480"     28.32  640  712  824  896   480  ???  ???  526
```

Restano i dati sulla durata dell'impulso di sincronismo verticale. Dal momento che la differenza rispetto all'esempio di riferimento non è molto grande, si può provare un po' a occhio, salvo verificare con la calcolatrice.

```
# 640x480 @ 60 Hz, 31.5 kHz hsync
Modeline "640x480"     28.32  640  712  824  896   480  491  494  526
```

Con questi valori, l'impulso di sincronismo dura 95 μs ((494 − 491) * 896 / 28320000), perfettamente accettabile.

Volendo verificare la frequenza orizzontale e verticale per sicurezza, si ottengono 31,58 kHz e 60,08 Hz, valori leggermente differenti rispetto a quelli di partenza, ma sicuramente tollerabili.

Rifiniture

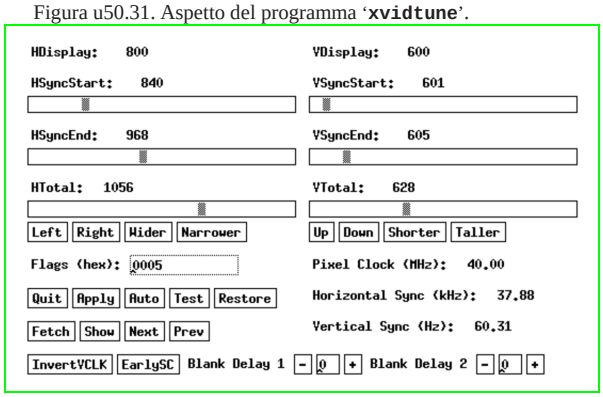
I valori che si calcolano a tavolino, non possono essere sempre perfetti al primo colpo. Se tutto va bene, può capitare che l'immagine appaia un po' troppo spostata rispetto al centro dello schermo. Di certo si possono utilizzare i controlli del monitor per spostarla, ma a volte non conviene esagerare, dovendo trovare un compromesso tra la visualizzazione di schermate a caratteri e l'uso di X.

Quello che bisogna fare è ritoccare le dimensioni degli impulsi di sincronismo, oltre a cercare la loro collocazione ideale. Per questo però, viene in aiuto un programma apposito, che permette di verificare al volo valori differenti. Si tratta di '**xvidtune**'.

Il programma '**xvidtune**'[2] permette di verificare la configurazione delle modalità video utilizzabili attraverso il servente X attivo. Generalmente viene avviato senza opzioni, ottenendo un funzionamento interattivo:

```
xvidtune [opzioni]
```

Figura u50.31. Aspetto del programma '**xvidtune**'.



Come si può osservare dalla figura, i controlli dal lato sinistro riguardano la scansione orizzontale, mentre quelli del lato destro quella verticale. In basso a destra si può tenere sotto controllo il valore della frequenza di dot-clock (*pixel clock*), della frequenza di sincronizzazione orizzontale e verticale.

Al posto di utilizzare le barre si scorrimento, si possono selezionare i pulsanti grafici corrispondenti all'azione che si vuole ottenere: LEFT dovrebbe spostare l'immagine a sinistra, RIGHT a destra, WIDER dovrebbe allargare l'immagine, e NARROWER dovrebbe restringerla.

Per verificare l'effetto delle modifiche, basta selezionare il pulsante grafico TEST.

I pulsanti grafici NEXT e PREV permettono di passare alla modalità grafica successiva (quella che si otterrebbe con la combinazione [*Ctrl Alt* +]) e precedente ([*Ctrl Alt* -]).

Altri programmi affini

- *modeline(1)* [3]

  Si tratta di un programma che calcola i valori da associare alla direttiva '**Modeline**', oppure '**Mode**', dei programmi che usano la grafica e devono sapere come gestire la scansione dell'immagine.

Riferimenti

- Eric S. Raymond, *XFree86 Video Timings HOWTO*
  http://www.linux.org/docs/ldp/howto/HOWTO-INDEX/howtos.html

[1] **Read-edid** software libero per la maggior parte GNU GPL

[2] **X** MIT più altre licenze per porzioni particolari di codice

[3] **modeline** GNU GPL

Posta elettronica

Sendmail: introduzione 279
Destinatari e formati degli indirizzi 279
Alias, inclusione e forward 280
Configurazione di Sendmail con il pacchetto di Berkeley 281
Esempio di una distribuzione GNU/Linux 284
Exim 3: introduzione 287
Compatibilità con Sendmail e differenze importanti 287
Installazione 288
Organizzazione della configurazione 290
Elementi comuni della configurazione 291
Configurazione in pratica 294
Avvio di Exim 299
Code e registri 301
Riferimenti 302
Ssmtp 303
Configurazione 303
Ricezione della posta elettronica 304
Considerazioni sulla sicurezza 304

Sendmail: introduzione

Destinatari e formati degli indirizzi 279
Alias, inclusione e forward 280
Configurazione di Sendmail con il pacchetto di Berkeley 281
Introduzione al sistema 281
Struttura e contenuto del file di configurazione 282
Macro «VERSIONID» 282
Macro «OSTYPE» 283
Macro «DOMAINS» 283
Macro «MAILERS» 283
Macro «FEATURE» 284
Macro «HACK» 284
Esempio di una distribuzione GNU/Linux 284
File «/etc/mail/ip_allow» 285
File «/etc/mail/name_allow» 285
File «/etc/mail/relay_allow» 285
File «/etc/mail/deny» 286

software non libero: non è consentita la commercializzazione a scopo di lucro

Sendmail [1] è divenuto lo standard per quanto riguarda i programmi di gestione della posta elettronica in qualità di MTA. La sua adattabilità e la conseguente difficoltà nella definizione della sua configurazione, sono estreme.

Nel capitolo ??capitolo email?? si è già accennato al funzionamento di Sendmail. Questo capitolo espande un po' i concetti, ma si tratta sempre di informazioni limitate; il documento di riferimento per questo resta: *Sendmail* edito da O'Reilly.

Destinatari e formati degli indirizzi

Sendmail, per quanto riguarda la composizione degli indirizzi di posta elettronica, utilizza le convenzioni seguenti.

- Ciò che appare tra parentesi viene eliminato, perché considerato un commento.
- Ciò che appare tra parentesi angolari ('**<>**') viene preferito rispetto a ogni altra indicazione. In pratica, ciò permette di comporre gli indirizzi inserendo anche il nome effettivo del mittente o del destinatario, evidenziando l'indirizzo di posta elettronica vero e proprio all'interno delle parentesi angolari. Per esempio,

  '**`Tizio Tizi <tizio@dinkel.brot.dg>`**'

  è un modo formalmente corretto per abbinare all'indirizzo *tizio@dinkel.brot.dg* il nome e cognome dell'utente: Tizio Tizi.
- Gli apici doppi permettono di delimitare una stringa. In questo modo, alle volte si delimita il nominativo dell'utente, come nell'esempio seguente:

  '**`"Tizio Tizi" <tizio@dinkel.brot.dg>`**'

  Nello stesso modo, la barra obliqua inversa ('`\`') può essere usata per proteggere il carattere successivo.

Per Sendmail, il destinatario di un messaggio di posta elettronica può essere anche un file o un programma. In pratica, se l'indirizzo utilizzato inizia con una barra verticale ('`|`'), si intende trattarsi di un condotto, all'interno del quale deve essere inviato il messaggio; se invece l'indirizzo inizia con una barra obliqua normale ('`/`'), si intende trattarsi di un file, che viene creato appositamente oppure gli viene aggiunto il testo del messaggio se esiste già.

L'utilizzo di questi indirizzi speciali, riferiti a file o a condotti, può essere fatto ovunque; per esempio nel file '`~/.forward`' o come

destinatario di un alias nel file '`/etc/aliases`'. Queste possibilità, tra le altre cose, sono alla base del funzionamento delle liste di posta elettronica (*mailing-list*).

Alias, inclusione e forward

All'interno di un sistema è possibile definire dei recapiti fittizi, definiti alias. La predisposizione di questi viene fatta nel file '`/etc/aliases`', ma prima che questi abbinamenti siano recepiti da Sendmail, occorre rigenerare il file '`/etc/aliases.db`' con il comando '**`newaliases`**'. Attraverso gli alias è possibile:

- definire dei nominativi utenti lunghi e articolati, che non sarebbero ammissibili nella gestione normale di un sistema Unix (il quale pone generalmente il limite degli otto caratteri di lunghezza per i nomi degli utenti),
- definire dei nominativi di utenti standard riferiti a determinate competenze amministrative tipiche, girando i messaggi loro rivolti alle persone che ricoprono effettivamente gli incarichi corrispondenti;
- definire un elenco di destinatari differenti a cui deve essere inviata una copia dei messaggi riferiti a un certo alias;
- definire un condotto contenente un comando che deve occuparsi di elaborare i messaggi riferiti a un certo alias;
- definire un file per l'archiviazione dei messaggi indirizzati a un certo alias;
- definire un elenco di destinatari differenti in base a un elenco di indirizzi contenuto in un file esterno.

L'esempio seguente fa in modo che i messaggi inviati all'utente fittizio '**`Tizio.Tizi`**' siano girati al nome dell'utente gestito effettivamente nel sistema.

```
Tizio.Tizi:     tizio
```

L'esempio seguente riguarda la situazione tipica in cui i messaggi indirizzati a un utente fittizio riferito a una competenza amministrativa vengono girati all'utente reale che svolge quel compito particolare.

```
postmaster:     daniele
```

L'esempio seguente mostra un alias per il quale i messaggi vengono rinviati (vengono fatti proseguire, o meglio, secondo la tradizione postale, vengono proseguiti) e duplicati per una serie di utenti che devono essere informati contemporaneamente.

```
abuse:          daniele, tizio, caio@roggen.brot.dg
```

L'esempio seguente mostra un alias per il quale tutti i messaggi vengono elaborati da un comando, che li riceve attraverso lo standard input. Questo è il modo tipico attraverso cui si inviano i messaggi a un programma di gestione di una lista di posta elettronica (*mailing-list*).

```
lista-pippo:    "| /home/liste/bin/ricezione-messaggi lista-pippo"
```

L'inclusione è un modo di definire un alias dinamico, riferito a un elenco di indirizzi contenuti in un file di testo normale. La forma

```
:include:percorso_assoluto
```

equivale a includere tutti gli indirizzi definiti nel file specificato, che deve comprendere necessariamente il percorso assoluto per raggiungerlo. Utilizzando questa forma di definizione degli elenchi di destinatari, si evita di dover modificare ogni volta il file '`/etc/aliases`', ma soprattutto si evita di dover rieseguire il comando '**`newaliases`**'.

L'esempio seguente invia i messaggi destinati all'utente fittizio '**`lista-pippo-inv`**' a tutto l'elenco contenuto nel file '`/home/liste/pippo/iscritti`'.

```
lista-pippo-inv:        :include:/home/liste/pippo/iscritti
```

Il *forward* è la gestione di un alias personale (allo scopo di fare proseguire i messaggi verso altre destinazioni), che ogni utente può definire senza dover chiedere la modifica del file '`/etc/aliases`'. Si

possono fare proseguire i messaggi generando il file di testo '`~/.forward`' che può contenere uno o più indirizzi differenti, compresi i condotti, i file e le inclusioni. Il risultato che si ottiene è che i messaggi destinati all'utente che ha predisposto questo file nella propria directory personale, vengono rinviati a tutti gli indirizzi contenuti nel file stesso. Generalmente, per la sua natura, il file '`~/.forward`' viene usato dagli utenti che hanno diversi recapiti e vogliono concentrare la posta elettronica in un solo punto di destinazione. Per questo motivo, nel file '`~/.forward`' viene indicato quasi sempre un solo indirizzo di posta elettronica.

Configurazione di Sendmail con il pacchetto di Berkeley

Si è già accennato al fatto che la configurazione di Sendmail, attraverso la modifica diretta del file '`/etc/sendmail.cf`', sia un'impresa estrema. Fortunatamente, per alleviare queste difficoltà, si sono sviluppati nel tempo diversi programmi in grado di generare automaticamente il file '`/etc/sendmail.cf`' utilizzando dei segmenti di codice già pronto da combinare opportunamente assieme.

Attualmente, il tipo di configurazione più diffuso è quello predisposto dall'università di Berkeley. Si tratta di una serie di file macro per M4, un macro-compilatore concettualmente analogo al precompilatore del linguaggio C.

Il pacchetto viene installato da qualche parte, a seconda dell'organizzazione predisposta dalla propria distribuzione GNU, ma probabilmente si tratta della directory '`/usr/lib/sendmail-cf/`'. Da quella directory se ne diramano altre contenenti i diversi pezzi di configurazione che possono essere combinati assieme.

Introduzione al sistema

A partire dalla directory di origine del pacchetto di configurazione di Sendmail, si trovano in particolare i file *readme* che rappresentano tutta la documentazione disponibile, oltre a una serie di directory contenenti a loro volta i file componenti del sistema di macro.

Directory	Descrizione
'`m4/`'	Contiene alcune macro di partenza, di cui, la più importante è '`cf.m4`' che viene usata per iniziare il procedimento.
'`cf/`'	Contiene i file di configurazione utilizzati dalla macro '`m4/cf.m4`'; questi file hanno l'estensione '`.mc`'. Di questi file ne viene usato solo uno: quello predisposto per il proprio sistema. È molto probabile che la propria distribuzione GNU inserisca il file utilizzato effettivamente per ottenere la configurazione di Sendmail che questa utilizza; potrebbe trattarsi di '`linux.mc`' oppure di un altro nome che ricorda quello della distribuzione (per esempio '`redhat.mc`').
'`sh/`'	Contiene degli script di shell utilizzati automaticamente da M4, in base alle istruzioni contenute nelle macro utilizzate.
altre directory	Le altre directory che discendono dall'origine del pacchetto di configurazione, sono utilizzate per classificare i vari file macro incorporabili in quello che si scrive all'interno della directory '`cf/`'. Per esempio, un'istruzione del tipo '**`MAILER(local)`**', fa riferimento al file '`mailer/local.m4`'.

Quando si predispone un file di configurazione nella directory '`cf/`', la sua compilazione avviene nel modo seguente:

```
m4 ../m4/cf.m4 file_di_configurazione > file-risultato
```

Per esempio, supponendo di avere realizzato il file di configurazione '`cf/prova.mc`' e di voler generare il file '`cf/prova.cf`', si procede come segue:

```
# cd /usr/lib/sendmail-cf [Invio]
```

In questo modo ci si posiziona nella directory principale del pacchetto di configurazione.

```
# cd cf [Invio]
```

Prima di iniziare la compilazione occorre posizionarsi nella directory contenente il file di configurazione.

```
# m4 ../m4/cf.m4 prova.mc > prova.cf [Invio]
```

A questo punto il file '`cf/prova.cf`' è stato generato, quindi è sufficiente cambiargli nome e sostituirlo al posto del vecchio '`/etc/sendmail.cf`'.

Naturalmente, perché Sendmail prenda atto della nuova configurazione, deve essere riavviato (dovrebbe bastare l'invio di un segnale di aggancio, '**`SIGHUP`**').

Struttura e contenuto del file di configurazione

Il file di configurazione inizia generalmente con delle annotazioni, che possono riguardare il copyright o lo scopo del file. Osservando i file già esistenti si potrebbe pensare che il simbolo '`#`' rappresenti l'inizio di un commento; in realtà si tratta di un commento per il file '`.cf`' che si vuole generare, perché all'interno del sistema di macro di M4 è stato ridefinito opportunamente il simbolo di commento in modo che '`#`' venga trattato come un carattere qualunque senza significati particolari. Questo significa che le espansioni hanno luogo anche all'interno dei commenti per il file '`/etc/sendmail.cf`'.

Il modo adottato comunemente per eliminare le intestazioni contenenti le informazioni sul copyright e le riserve all'uso dei vari file, è quello di dirigere l'output in modo da perderlo, attraverso la macro '**`divert(-1)`**'.

In teorica, l'aspetto normale di un file di configurazione per questo pacchetto dovrebbe essere il seguente:

```
divert(-1)
#
# Copyright (c) 1983 Eric P. Allman
# Copyright (c) 1988, 1993
#       The Regents of the University of California.  All rights reserved.
#
# Redistribution and use in source and binary forms, with or without...
# ...
divert(0)dnl
include('../m4/cf.m4')
VERSIONID('@(#)generic-linux.mc 8.3 (Berkeley) 3/23/96')
OSTYPE(linux)dnl
DOMAIN(generic)dnl
MAILER(local)dnl
MAILER(smtp)dnl
```

In pratica, questo potrebbe generare un file '`.cf`' insufficiente al funzionamento corretto di Sendmail.

Si può osservare all'inizio l'inclusione del file '`m4/cf.m4`' che è il responsabile dell'impostazione di questo sistema di macro.

Quasi tutte le macro specifiche che si utilizzano in questo file (quelle che appaiono in lettere maiuscole), rappresentano in realtà l'inclusione di un file, quello che appare come parametro, proveniente dalla directory corrispondente al nome della macro stessa. Per esempio, '**`OSTYPE(linux)`**' rappresenta in pratica l'inclusione del file '`ostype/linux.m4`'. Nelle sezioni seguenti vengono descritte brevemente alcune di queste macro specifiche.

Macro «VERSIONID»

```
VERSIONID(descrizione_della_versione)
```

La macro '**`VERSIONID`**' permette semplicemente di includere un'annotazione sulla versione della configurazione, nei commenti del file '`.cf`' generato. È utile per documentare diversi tipi di configurazione, tenuto conto che la forma per definire la versione non è prestabilita.

Macro «OSTYPE»

```
OSTYPE(macro_da_includere)
```

Attraverso la macro '**`OSTYPE`**' si può definire il nome del sistema operativo utilizzato. In pratica, si tratta di indicare il nome (senza estensione) di un file macro contenuto nella directory '`ostype/`', da includere in quel punto.

Attraverso l'inclusione di questo file, si ottiene la definizione di alcune informazioni importanti riguardo all'installazione di Sendmail nel proprio sistema operativo; per esempio si può definire la collocazione del file contenente gli alias, il programma da usare per la consegna dei messaggi, le opzioni e gli argomenti che questo programma deve avere. Tutte queste informazioni vengono specificate attraverso la definizione di macro specifiche, come se si trattasse della definizione di variabili. Se tali macro non sono definite in questa occasione, vengono definite in un altro momento, ricevendo un valore predefinito, come documentato regolarmente nei file che accompagnano il pacchetto di configurazione.

L'esempio più semplice possibile del file '`ostype/linux.m4`' è il seguente,

```
divert(-1)
#
# ...

divert(0)
define('LOCAL_MAILER_PATH', /bin/mail)dnl
```

dove si definisce soltanto che il programma di consegna dei messaggi è '`/bin/mail`'. In pratica però, normalmente, questo file viene modificato opportunamente da chi allestisce il pacchetto di configurazione per una distribuzione GNU particolare.

La possibilità che questo file non sia conforme alla distribuzione standard del pacchetto di configurazione di Sendmail, deve essere tenuto in considerazione quando si vuole provare a generare un file '`.cf`' differente dal '`/etc/sendmail.cf`' già predisposto dalla propria distribuzione. Infatti, le modifiche che potrebbero essere state apportate possono pregiudicare l'effetto prevedibile delle altre macro.

Macro «DOMAINS»

```
DOMAINS(macro_da_includere)
```

Attraverso la macro '**`DOMAINS`**' si può definire il nome di una configurazione riferita a un dominio particolare. Si ottiene in pratica l'inclusione di un file contenuto nella directory '`domains/`'.

Il pacchetto di configurazione fornisce il file '`domains/generic.m4`', che dovrebbe adattarsi a tutte le situazioni normali. Spesso, questo non viene utilizzato, inserendo direttamente quello che serve nel file di configurazione normale.

Quello che segue è un estratto dal file '`domains/generic.m4`'.

```
divert(-1)
#
# ...

divert(0)
VERSIONID('@(#)generic.m4       8.3 (Berkeley) 3/24/96')
define('confFORWARD_PATH', '$z/.forward.$w:$z/.forward')dnl
FEATURE(redirect)dnl
FEATURE(use_cw_file)dnl
```

Macro «MAILERS»

```
MAILERS(macro_da_includere)
```

Attraverso la macro '**`MAILERS`**' si può definire il nome di una configurazione riferita a un tipo particolare di sistema di invio dei messaggi. Si ottiene in pratica l'inclusione di un file contenuto nella directory '`mailers/`'.

Normalmente, questa macro viene utilizzata più volte all'interno del file di configurazione, per definire diverse possibilità. Tipicamente si tratta di:

```
MAILER(local)
```

che si occupa della gestione dei messaggi all'interno del sistema e viene utilizzato in modo predefinito;

```
MAILER(smtp)
```

che si occupa di configurare la gestione dei messaggi attraverso il protocollo SMTP, cioè riguarda la configurazione necessaria all'invio dei messaggi al di fuori del sistema.

Nel primo caso si ha l'inclusione del file '`mailers/local.m4`', nel secondo di '`mailers/smtp.m4`'

Dalla macro '**`MAILER(smtp)`**' dipende la base del sistema di sicurezza contro gli utilizzi indesiderati del proprio servente SMTP. Infatti, è qui che vengono definite le istruzioni necessarie nel file '`.cf`' per impedire l'utilizzo da parte di nodi che non facciano parte della zona DNS di competenza. Cioè, quello che si vuole evitare è che un nodo diverso da quelli definiti nella zona per cui è stato previsto un record '**`MX`**', possa utilizzare il servente SMTP per raggiungere indirizzi al di fuori del sistema locale (si veda eventualmente quanto discusso nel capitolo precedente).

Macro «FEATURE»

```
FEATURE( macro_da_includere )
```

Attraverso la macro '**`FEATURE`**' si può definire il nome di una configurazione riferita a una particolarità che si vuole includere. Si ottiene in pratica l'inclusione di un file contenuto nella directory '`feature/`'.

Normalmente, questa macro viene utilizzata più volte all'interno del file di configurazione, ma questo preferibilmente prima di '**`MAILER`**'.

Macro «HACK»

```
HACK( macro_da_includere )
```

Attraverso la macro '**`HACK`**' si può definire il nome di una configurazione riferita a una particolarità sperimentale che si vuole includere. Si ottiene in pratica l'inclusione di un file contenuto nella directory '`hack/`'.

Teoricamente, questa macro non dovrebbe essere utilizzata; in pratica succede spesso il contrario a causa delle esigenze di definire dei filtri aggiuntivi contro gli accessi indesiderati.

Esempio di una distribuzione GNU/Linux

A titolo di esempio, viene presentata la configurazione utilizzata dalla distribuzione GNU/Linux Red Hat 5.0, trattandosi precisamente del file '`cf/redhat.mc`'.

```
divert(-1)
include('../m4/cf.m4')
define('confDEF_USER_ID',''8:12'')
OSTYPE('linux')
undefine('UUCP_RELAY')
undefine('BITNET_RELAY')
FEATURE(redirect)
FEATURE(always_add_domain)
FEATURE(use_cw_file)
FEATURE(local_procmail)
MAILER(procmail)
MAILER(smtp)
HACK(check_mail3,'hash -a@JUNK /etc/mail/deny')
HACK(use_ip,'/etc/mail/ip_allow')
HACK(use_names,'/etc/mail/name_allow')
HACK(use_relayto,'/etc/mail/relay_allow')
HACK(check_rcpt4)
HACK(check_relay3)
```

La prima cosa che si osserva è che il file inizia con la macro '**`divert(-1)`**', senza commenti da eliminare e senza il consueto '**`divert(0)`**' successivo. In questo modo, dal momento che nessuna delle macro utilizzate dopo deve restituire qualcosa, si evita di terminare le varie macro con il solito '**`dnl`**'.

Per sicurezza, nel caso servisse, vengono cancellate le macro '**`UUCP_RELAY`**' e '**`BITNET_RELAY`**'.

Invece di utilizzare una macro '**`DOMAIN`**' vengono incluse direttamente le particolarità attraverso l'uso della macro '**`FEATURE`**'. In particolare viene definito quanto segue.

- '**`FEATURE(redirect)`**'

  Si tratta di una particolarità poco importante, con la quale si ottiene di emettere un avviso nel caso sia utilizzato un indirizzo di posta elettronica nella forma '***indirizzo_normale*`.REDIRECT`**'. Si ottiene una segnalazione di errore in cui si invita a utilizzare la parte di indirizzo precedente a '**`.REDIRECT`**'.

- '**`FEATURE(always_add_domain)`**'

  Inserendo questa configurazione, si ottiene di aggiungere il nome a dominio all'utente destinatario quando questo non viene specificato esplicitamente.

- '**`FEATURE(use_cw_file)`**'

  In questo modo si ottiene di fare accettare a Sendmail l'identificazione del proprio nodo attraverso uno dei nomi elencati nel file '`/etc/sendmail.cw`'.

- '**`FEATURE(local_procmail)`**'

  Fa in modo di utilizzare '**`procmail`**' come sistema di consegna dei messaggi in ambito locale.

Le macro '**`HACK`**' inserite alla fine, sono state aggiunte per permettere una migliore gestione dei filtri di accesso al servizio di invio dei messaggi, comprendendo in questo anche la definizione di nodi per i quali il proprio servente SMTP può agire come relè.

Per la precisione, è consentito l'uso dei file descritti nelle sezioni seguenti.

File «/etc/mail/ip_allow»

Si tratta di un file di testo contenente un elenco di indirizzi IP (uno per riga) riferiti a nodi particolari o a intere reti. A questi elaboratori viene consentito di utilizzare il servente SMTP come relè. Per esempio,

```
192.168.1.2
192.168.2
```

permette l'accesso al nodo 192.168.1.2 e a tutta la rete 192.168.2.

Questo file, al di fuori della configurazione particolare della distribuzione Red Hat, potrebbe chiamarsi '`/etc/mail/LocalIP`'.

File «/etc/mail/name_allow»

Si tratta di un file di testo contenente un elenco di nomi a dominio (uno per riga) riferiti a nodi particolari o a tutti i nodi di un dominio particolare. A questi elaboratori viene consentito di utilizzare il servente SMTP come relè. Per esempio,

```
roggen.brot.dg
mehl.dg
```

permette l'accesso al nodo *roggen.brot.dg* e a tutto il dominio *mehl.dg*.

Questo file, al di fuori della configurazione particolare della distribuzione Red Hat, potrebbe chiamarsi '`/etc/mail/LocalNames`'.

File «/etc/mail/relay_allow»

Si tratta di un file di testo contenente un elenco di indirizzi IP o di nomi a dominio (uno per riga) riferiti a nodi particolari o a tutti i nodi di una rete particolare o di un dominio. Questi indirizzi sono ammessi come destinatari di messaggi quando il servente SMTP viene utilizzato come relè. Per esempio,

```
192.168
roggen.brot.dg
mehl.dg
```

permette di inviare messaggi alla rete 192.168.∗.∗, al nodo *roggen.brot.dg* e a tutto il dominio *mehl.dg*, quando il servente SMTP funziona come relè.

Questo file, al di fuori della configurazione particolare della distribuzione GNU/Linux Red Hat, potrebbe chiamarsi '`/etc/mail/RelayTo`'.

File «/etc/mail/deny»

Il file di testo '`/etc/mail/deny`' viene utilizzato per annotare un elenco di indirizzi di posta elettronica, nomi a dominio e indirizzi IP di mittenti indesiderati. A fianco di ogni indirizzo, separato da un carattere di tabulazione (<*HT*>), si indica il messaggio di errore che si vuole sia restituito all'MTA che ha contattato il servente per l'inoltro del messaggio.

Segue un esempio molto semplice di questo file.

```
spam@marameo.dg "Spiacente sig. Spam, non accettiamo messaggi da Lei."
spam.brot.dg    "Dal Vostro host non accettiamo email."
spammer.dg      "Non vogliamo spam, grazie."
192.168.13.13   "Dal Vostro host non accettiamo email."
192.168.17      "Non vogliamo spam, grazie."
```

Questo file non può essere usato così com'è; occorre generare un file adatto a Sendmail. Si utilizza in pratica il comando seguente:

```
# makemap -v hash /etc/mail/deny < /etc/mail/deny [Invio]
```

Quello che si ottiene è il file '`/etc/mail/deny.db`'.

[1] **Sendmail** software non libero: non è consentita la commercializzazione a scopo di lucro

Exim 3: introduzione

Compatibilità con Sendmail e differenze importanti 287
Installazione 288
- Utente specifico 288
- File «/etc/aliases» 289
- File «~/.forward» 289
- Directory di destinazione dei messaggi locali 289

Organizzazione della configurazione 290
- Struttura 290

Elementi comuni della configurazione 291
- Macro 291
- Assegnamento 291
- Valori booleani 291
- Interi 292
- Numeri con parte decimale 292
- Intervalli orari 292
- Stringhe 292
- Elenchi di stringhe 293
- Espansione delle stringhe 293
- Espressioni regolari 293

Configurazione in pratica 294
- Configurazione principale 294
- Configurazione dei driver 297
- Configurazione dei tentativi ripetuti 297
- Configurazione della riscrittura degli indirizzi 298
- Configurazione generica 298
- Interazione con Procmail 298

Avvio di Exim 299
Code e registri 301
- Archiviazione dei file delle registrazioni 302

Riferimenti 302

In questo capitolo si introduce l'utilizzo di Exim, [1] un MTA che offre qualche piccolo vantaggio rispetto all'uso di Sendmail: è abbastanza compatibile con le consuetudini di questo ultimo; ha un sistema di configurazione meno criptico; è predisposto per IPv6; quando possibile utilizza processi senza i privilegi dell'utente '**`root`**', in modo da lasciare meno occasioni alle aggressioni.

Questo capitolo si riferisce precisamente a Exim nelle versioni 3.∗. Le versioni successive prevedono una configurazione differente.

Compatibilità con Sendmail e differenze importanti

Exim è compatibile con Sendmail per tutti quegli aspetti che coinvolgono gli utenti comuni e anche per ciò che riguarda gli amministratori che non hanno o non desiderano avere conoscenze troppo approfondite sulla gestione della posta elettronica. Questa compatibilità riguarda tre punti fondamentali: il file '`/etc/aliases`', i file '`~/.forward`' e un collegamento che simula la presenza dell'eseguibile '**`sendmail`**'. Per di più, l'eseguibile '**`exim`**' accetta buona parte delle opzioni standard di '**`sendmail`**', in modo da permettere il funzionamento di programmi come Mailx, o il funzionamento di script che si affidano alla presenza dell'eseguibile '**`sendmail`**'.

I file ‘`/etc/aliases`’ e ‘`~/.forward`’ si comportano in modo quasi identico rispetto a quando è in funzione Sendmail. In particolare, con ‘`~/.forward`’ si possono usare anche delle estensioni.

Un’eccezione, rispetto alla compatibilità di questi file, riguarda l’indicazione di condotti. Con Sendmail, si presume che il comando sia elaborato da una shell; con Exim, no. Di conseguenza, i comandi interni di questa non sono accessibili. Si osservino gli esempi seguenti, riferiti al contenuto del file ‘`/etc/aliases`’: si tratta dello stesso condotto a cui vengono ridiretti i messaggi giunti per l’utente ipotetico, denominato ‘**`lista-pippo`**’.

```
# con Sendmail
lista-pippo:  "| exec /home/liste/bin/ricezione-messaggi lista-pippo"
```

```
# con Sendmail senza exec
lista-pippo:  "| /home/liste/bin/ricezione-messaggi lista-pippo"
```

```
# con Exim non si può usare exec che è un comando interno di shell
lista-pippo:  "| /home/liste/bin/ricezione-messaggi lista-pippo"
```

```
# con Exim, avviando prima una shell e quindi il comando
lista-pippo:  "| /bin/sh -c '/home/liste/bin/ricezione-messaggi lista-pippo'"
```

Se si deve inserire un condotto in un file ‘`~/.forward`’, vale lo stesso ragionamento, con la differenza che qui non si mette più l’indicazione del destinatario perché è implicita (ma questo vale anche per Sendmail).

Il primo vantaggio che si osserva rispetto a Sendmail è che il file ‘`/etc/aliases`’ non deve essere «ricompilato» attraverso ‘**`newaliases`**’: basta modificarlo e non occorre nemmeno riavviare il servizio perché viene riletto ogni volta dal sistema di consegna locale.

Se nel file ‘`~/.forward`’ si inserisce un indirizzo che crea un circolo senza fine, come per esempio quando si indica lo stesso indirizzo dell’utente per il quale è stato creato il file, i messaggi vengono consegnati presso quello stesso recapito, invece di ignorarli semplicemente.

I permessi del file ‘`~/.forward`’ non possono concedere la scrittura al gruppo e al resto degli utenti. Questo particolare va considerato quando si utilizza una maschera dei permessi pari a 002_8 (è così, solitamente, quando si usano i gruppi privati), che tende a concedere la scrittura al gruppo in modo predefinito.

Come già accennato, Exim fornisce un collegamento denominato ‘**`sendmail`**’, per favorire il funzionamento dei programmi che dipendono dalla presenza di questo; inoltre, offre il collegamento ‘**`mailq`**’, come fa Sendmail, per permettere la verifica dei messaggi in coda.

Installazione

L’installazione di Exim può costituire un problema se non si parte da un pacchetto già predisposto per la propria distribuzione GNU; quindi è decisamente preferibile cercare un tale pacchetto già pronto. Purtroppo, in certi casi, anche questo non basta: occorre preparare qualcosa prima, forse è necessario definire la configurazione; infine occorre fare delle sistemazioni finali dopo alcune prove di verifica.

Utente specifico

Quando possibile, se la configurazione lo consente, Exim cerca di avviare processi con privilegi inferiori a quelli dell’utente ‘**`root`**’. Per esempio, la consegna locale della posta avviene normalmente con un processo che utilizza i privilegi dell’utente destinatario. In tutte le altre circostanze, si può stabilire un utente e un gruppo che Exim deve utilizzare: nei sistemi che utilizzano i gruppi privati (un gruppo per ogni utente), si potrebbe creare l’utente e il gruppo ‘**`exim`**’; negli altri sistemi, può essere conveniente creare solo l’utente ‘**`exim`**’, a cui abbinare il gruppo ‘**`mail`**’.

Pertanto, la prima cosa da fare è la creazione di questo utente ed eventualmente del gruppo corrispondente (se non è già previsto, o

se si utilizzano i gruppi privati). Nel file ‘`/etc/passwd`’ potrebbe apparire una riga come quella seguente, dove il numero GID 12 è inteso corrispondere a ‘**`mail`**’.

```
exim:*:501:12:Exim mailer:/:
```

Se si usano i gruppi privati, si potrebbero avere i record seguenti, rispettivamente nei file ‘`/etc/passwd`’ e ‘`/etc/group`’.

```
exim:*:501:501:Exim mailer:/:
```

```
exim:x:501:
```

In ogni caso, come si è visto, è importante che l’accesso sia impossibile, cosa che si ottiene con l’asterisco nel campo della parola d’ordine.

File «/etc/aliases»

Dopo l’installazione di Exim, si può fare in modo di recuperare il vecchio ‘`/etc/aliases`’, se esiste, oppure se ne deve creare uno nuovo. Per il momento, fino a che non è stata vista la configurazione di Exim, è meglio lasciare stare gli alias che si traducono in file o in condotti.

Exim può essere configurato per utilizzare un file diverso da ‘`/etc/aliases`’ con questo stesso scopo, ma in generale dovrebbe essere conveniente mantenere la vecchia convenzione. In ogni caso, Exim ha bisogno della definizione di alcuni alias indispensabili: in generale, Exim non permette la consegna di messaggi direttamente all’utente ‘**`root`**’, quindi è necessario definire chi sia l’utente corrispondente che deve ricevere la posta diretta a ‘**`root`**’. Di seguito viene mostrato un esempio che dovrebbe andare bene in molte piattaforme GNU: l’alias di ‘**`root`**’ deve essere modificato opportunamente.

```
# Obbligatori
MAILER-DAEMON:  postmaster
abuse:          postmaster
postmaster:     root

# Ridirezione per evitare trucchi con gli utenti speciali di sistema.
bin:            root
daemon:         root
adm:            root
lp:             root
sync:           root
shutdown:       root
halt:           root
mail:           root
news:           root
uucp:           root
operator:       root
games:          root
gopher:         root
ftp:            root
nobody:         root
postgres:       root
exim:           root

# Chi è root (modificare in base alla realtà del proprio sistema)
#root:          daniele
```

File «~/.forward»

Anche la gestione dei file ‘`~/.forward`’ può essere controllata (e stravolta) attraverso la configurazione di Exim; tuttavia, se si desidera mantenere le consuetudini, si possono recuperare questi file che gli utenti potrebbero avere già utilizzato con Sendmail. Valgono naturalmente le stesse riserve già espresse in riferimento a destinatari costituiti da file o da condotti.

In ogni caso, perché tali file ‘`~/.forward`’ possano essere accettati da Exim, occorre che siano assenti i permessi di scrittura per il gruppo e per gli altri utenti. Utilizzando la shell Bash, si potrebbe usare un comando come quello seguente:

```
# find /home -name .forward -exec chmod go-w \{\} \;
```
[*Invio*]

Directory di destinazione dei messaggi locali

Sendmail utilizza una directory comune a tutti gli utenti per inserirvi i file contenenti i messaggi di questi, quando giungono a destinazione. In passato si è trattato di ‘`/var/spool/mail/`’ e attualmente dovrebbe essere ‘`/var/mail/`’, per conformità con lo standard

FHS. Exim può funzionare nello stesso modo, oppure può consegnare i messaggi direttamente nelle directory personali degli utenti. In generale, qualunque sia la scelta, è necessario che la variabile di ambiente '**MAIL**' contenga il percorso assoluto per raggiungere il file di destinazione, in modo che i programmi di lettura della posta vi si possano adeguare. Questo lo si fa normalmente nella definizione del profilo della shell personale.

Per esempio, se si utilizza la shell Bash, il file '`/etc/profile`' potrebbe contenere le righe seguenti per indicare che i file dei messaggi si trovano nella directory '`/var/mail/`'.

```
MAIL="/var/mail/$USER"
export MAIL
```

Nel caso la posta venisse consegnata nel file '`~/Messaggi`' della directory personale di ogni utente, l'istruzione per definire la variabile '**MAIL**' potrebbe essere la seguente:

```
MAIL="$HOME/Messaggi"
export MAIL
```

Infine, si deve tenere presente che Exim utilizza i privilegi dell'utente destinatario per aprire i file di destinazione, per cui i premessi della directory devono essere regolati convenientemente. Il problema si pone quando si usa la directory '`/var/mail/`', o un'altra simile, per tutti i file di destinazione: è necessario che sia attribuita a questa directory la modalità 1777_8. Infatti, l'attivazione del bit Sticky, permette il blocco dei file (*lock*). Se non si ha l'accortezza di sistemare questo particolare, la posta elettronica non può essere consegnata.

```
# chmod 1777 /var/mail [Invio]
```

Organizzazione della configurazione

La configurazione di Exim è più semplice di Sendmail, ma resta comunque una cosa piuttosto delicata, dati i problemi che sono coinvolti nella gestione della posta elettronica. Alla fine di questo gruppo di sezioni viene mostrato un esempio completo di configurazione che dovrebbe funzionare correttamente nella maggior parte delle situazioni.

La documentazione di Exim è voluminosa e abbastanza dettagliata. Questo è un fatto positivo; purtroppo occorre dedicarvi un po' di tempo per la sua lettura. Se si desidera utilizzare Exim a livello professionale, ciò diventa necessario.

Il file di configurazione di Exim, volendo seguire lo standard dei sistemi GNU (e non solo di quelli), dovrebbe trovarsi nella directory '`/etc/`'. In pratica però, potrebbe non essere così. Una volta installato il pacchetto di Exim, occorre cercare il file di configurazione.

Il file di configurazione deve appartenere all'utente '**root**', oppure all'utente specificato in fase di compilazione dei sorgenti di Exim, attraverso l'opzione '**EXIM_UID**'; inoltre non può essere accessibile in scrittura dal gruppo né dagli altri utenti.

Quando si avvia Exim, se il file di configurazione contiene errori sintattici, viene emesso un messaggio di errore attraverso lo standard error, specificando anche la riga in cui questo si trova. Dopo tale segnalazione, Exim termina di funzionare, per cui il servizio non viene avviato.

Struttura

Il file di configurazione si divide in sei parti che devono apparire nell'ordine previsto. Ognuna di queste termina con la parola chiave '**end**', posta da sola in una riga. Le varie parti sono elencate di seguito.

1. Configurazione principale. Si tratta di direttive in cui si assegnano dei valori a delle opzioni di funzionamento.
2. Configurazione dei driver di trasporto. Si tratta della definizione dei meccanismi attraverso cui i messaggi vengono recapitati alla destinazione, copiandoli all'interno dei file, o inserendoli nei condotti.
3. Configurazione dei driver di direzione (*director*). Si tratta dei processi di consegna all'interno dei domini locali, cosa che include la gestione degli alias e del proseguimento dei messaggi (*forward*).
4. Configurazione dei driver di instradamento. Si tratta dei processi di consegna a destinazioni remote, ovvero, quelle destinazioni che non sono classificate come appartenenti ai domini locali.
5. Configurazione delle regole per i tentativi ripetuti (*retry*).
6. Configurazione delle regole di riscrittura. Si tratta della definizione di modifiche sistematiche a elementi dell'intestazione dei messaggi.

In ogni parte della configurazione possono apparire dei commenti; questi sono introdotti dal simbolo '**#**' all'inizio della riga e conclusi dalla fine della riga stessa. Non sono ammessi commenti alla fine delle direttive; in pratica, i commenti possono apparire solo su righe apposite. Inoltre, le righe bianche e quelle vuote vengono ignorate come di consueto.

Elementi comuni della configurazione

Prima di affrontare la descrizione di alcune direttive importanti che possono essere usate nel file di configurazione, conviene conoscere alcune convenzioni comuni, o direttive particolari che coinvolgono tutto l'insieme della configurazione.

Macro

All'interno della prima parte del file di configurazione, quella che riguarda le definizioni generali, è possibile inserire delle direttive che dichiarano delle macro. Queste si distinguono perché devono avere l'iniziale maiuscola. In generale, per convenzione comune derivante da altri linguaggi di programmazione, le macro si dichiarano con nomi composti esclusivamente da lettere maiuscole.

```
nome = valore_da_sostituire
```

Il nome può essere composto da lettere, numeri e dal trattino basso ('_'); inoltre, come accennato, la prima lettera deve essere maiuscola. Il valore che si abbina a questo nome, è tutto ciò che appare dopo il simbolo '**=**', escludendo eventuali spazi iniziali, fino alla fine della riga.

Assegnamento

In molti punti del file di configurazione si usano delle direttive che rappresentano in pratica l'assegnamento di un valore a una sorta di variabile. La sintassi è semplice e intuitiva.

```
nome = valore
```

La differenza rispetto alla dichiarazione di macro sta nel fatto che i nomi utilizzati in questo caso sono prestabiliti e non iniziano mai con una lettera maiuscola.

Valori booleani

Quando una variabile è fatta per definire l'attivazione o la disattivazione di qualcosa, può ricevere solo i valori *Vero* o *Falso*, espressi attraverso le solite parole chiave: '**true**' o '**yes**' rappresenta il valore *Vero*; '**false**' o '**no**' rappresenta il valore *Falso*.

In particolare, in presenza di variabili di questo tipo, è possibile fare a meno di indicare espressamente l'assegnamento, lasciando intuire il valore predefinito derivante dal nome della variabile. In generale, comunque, sarebbe bene esplicitare l'intenzione, se si vogliono utilizzare tali variabili.

Interi

I numeri interi possono essere annotati utilizzando diverse basi di numerazione:

- un numero che inizia con il prefisso 0x... viene inteso essere espresso in base esadecimale;
- un numero che inizia con uno zero viene inteso essere espresso in base ottale;
- un numero che inizia con una cifra diversa da zero viene inteso essere espresso in base decimale.

Tali numeri interi possono essere seguiti dalla lettera '**K**' (maiuscola) o dalla lettera '**M**', intendendo esprimere un multiplo di 1024, o di 1024*1024 rispettivamente (kibi e mebi delle convenzioni che si usano in informatica).

Numeri con parte decimale

Un numero contenente una parte decimale può essere espresso solo utilizzando la numerazione a base 10 (base decimale), indicando le cifre della parte frazionaria dopo un punto. Sono consentite un massimo di tre cifre decimali dopo la parte intera.

Intervalli orari

Le indicazioni di valori riferiti a intervalli orari (periodi di tempo), fanno uso di una serie di suffissi che descrivono il significato del numero che li precede.

- '**s**' secondi;
- '**m**' minuti;
- '**h**' ore;
- '**d**' giorni;
- '**w**' settimane.

Per esempio, il valore '**3h45m**' rappresenta 3 ore e 45 minuti. Questo formato di rappresentazione viene usato anche nell'output.

Stringhe

Le stringhe possono essere rappresentate con o senza apici doppi di delimitazione. L'utilizzo o meno di tale delimitazione ha delle conseguenze diverse.

Le stringhe non delimitate sono rappresentate da tutto ciò che appare dopo il simbolo '**=**' utilizzato nell'assegnamento, letteralmente, escludendo eventuali spazi iniziali, continuando fino alla fine della riga. Ciò significa in pratica che una stringa di questo tipo non può continuare nella riga successiva.

Si utilizzano le stringhe delimitate tutte le volte in cui occorre rappresentare qualcosa di particolare, come dei caratteri speciali, attraverso il prefisso '\' che assume il ruolo di carattere di escape, oppure quando è necessario continuare la stringa nella riga successiva.

La tabella u52.11 mostra l'uso di queste sequenze di escape ottenute con la barra obliqua inversa.

Tabella u52.11. Elenco delle sequenze di escape utilizzabili all'interno delle stringhe delimitate da apici doppi.

Simbolo	Significato
`\\`	Rappresenta una barra obliqua inversa singola.
`\n`	Il carattere <*NL*>.
`\r`	Il carattere <*CR*>.
`\t`	Una tabulazione orizzontale (<*HT*>).
`\`***n_ottale***	Identifica un carattere attraverso il suo numero ottale.
`\x`***n_esadecimale***	Identifica un carattere attraverso il suo numero esadecimale.

Inoltre, una barra obliqua inversa posta alla fine della riga, subito prima del codice di interruzione di riga, rappresenta la continuazione della stringa nella riga successiva, eliminando gli spazi iniziali aggiunti nella riga successiva.

Se si utilizza una barra obliqua inversa davanti a un carattere con il quale non forma alcuna sequenza di escape prevista, si conferma semplicemente il carattere successivo alla barra. Dal momento che all'interno delle stringhe possono essere usati altri simboli con significati speciali, si può usare la barra obliqua inversa per dare loro un significato puramente letterale.

Elenchi di stringhe

In situazioni determinate si può indicare un elenco di stringhe. La rappresentazione di tali elenchi avviene di fatto in una sola stringa, i cui elementi sono separati attraverso due punti verticali ('**:**'). Per esempio, '**uno:due:tre**' è un elenco composto dalle sottostringhe '**uno**', '**due**' e '**tre**'. È importante sapere subito che attorno ai due punti verticali, possono essere inseriti degli spazi, che poi vengono eliminati dalle sottostringhe; quindi, tornando all'esempio già presentato, sarebbe stata esattamente la stessa cosa scrivere '**uno: due :tre**'

A seconda delle esigenze, tali elenchi possono essere racchiusi globalmente attraverso gli apici doppi delle stringhe normali, oppure possono farne senza, con le stesse considerazioni già fatte su questo argomento.

Da quanto descritto, si intende che i due punti verticali abbiano un significato speciale, per cui non possono essere usati per altri scopi, a meno che questi appaiano in coppia ('**::**'), perché in tal caso rappresentano esattamente due punti verticali testuali.

Gli elenchi di stringhe vengono usati per rappresentare vari tipi di informazioni, spesso molto utili per una configurazione efficace di Exim. Qui viene trascurata la descrizione di queste indicazioni, che possono essere approfondite leggendo la documentazione originale.

Espansione delle stringhe

All'interno delle stringhe possono essere inseriti degli elementi che vengono sostituiti in qualche modo, in base a ciò che questi rappresentano. Per identificare tali elementi si utilizza il simbolo dollaro ('**$**') seguito da un nome, che eventualmente può anche essere racchiuso tra parentesi graffe, in caso si temano delle ambiguità.

$***nome_di_variabile***	${***nome_di_variabile***}

Esistono poi una serie di operazioni che possono essere compiute attraverso l'operatore di sostituzione (il dollaro), che qui non vengono descritte.

Nel caso si debba inserire il simbolo '**$**' in una stringa con un significato letterale, occorre indicare '**\$**' se si tratta di una stringa non delimitata, oppure '**\\$**' se si tratta di una stringa delimitata (incoerente, ma è così).

Espressioni regolari

In alcune situazioni, le stringhe possono servire a esprimere delle espressioni regolari. Tali espressioni regolari si distinguono per il fatto che iniziano con l'accento circonflesso ('**^**') e possono terminare o meno con il simbolo dollaro, che in tal caso rappresenta la fine della stringa con cui avviene il confronto.

Le regole per la realizzazione di tali espressioni regolari sono simili a quelle di Perl 5, facendo attenzione però alle barre oblique inverse, che se si trovano racchiuse tra apici doppi, devono essere raddoppiate.

Configurazione in pratica

Nelle sezioni successive vengono descritte, a gruppi di competenza, le direttive principali per la configurazione di Exim.

Configurazione principale

La prima parte del file di configurazione, fino al primo '**end**', riguarda la definizione delle opzioni principali. Come è già stato accennato, è in questa parte che possono essere create delle macro; la loro definizione si distingue in quanto i nomi di queste devono iniziare con una lettera maiuscola.

Questa parte della configurazione è la più semplice, perché richiede solo l'assegnamento di qualche valore a delle variabili prestabilite. L'elenco di tali variabili è molto lungo, ma in ogni caso, è sufficiente definire gli assegnamenti riferiti alle opzioni che si vogliono modificare rispetto a quanto risulta predefinito.

Directory	Descrizione
`exim_path =` ***percorso_assoluto_di_exim***	Permette di specificare il percorso assoluto dell'eseguibile '**exim**'. Questa informazione serve quando la collocazione del programma non corrisponde all'informazione indicata in fase di compilazione. La conoscenza di tale percorso serve a Exim quando deve avviare una copia di se stesso.
`primary_hostname =` ***nome_canonico***	Permette di definire esplicitamente il nome canonico primario del nodo locale. Se non viene specificata questa opzione, tale nome viene ottenuto dal sistema operativo, attraverso la funzione '**uname()**' (praticamente ciò che si otterrebbe con il comando '**hostname -f**').
`qualify_domain =` ***nome_di_dominio***	Permette di specificare il nome a dominio da aggiungere agli indirizzi che non lo possiedono. In mancanza di questa indicazione si usa il valore fissato con la direttiva '**primary_hostname**'.
`host_lookup =` ***indirizzo_ip/n_bit_maschera***	Permette di definire un gruppo di indirizzi per i quali verificare sempre il nome attraverso il DNS. Generalmente, viene assegnato '*****' che rappresenta ogni indirizzo possibile.

Directory	Descrizione
`local_domains =` ***nome_locale*** `[:` ***nome_locale*** `]…`	Permette di definire i nomi a dominio completi che fanno capo al sistema locale, per i quali la posta elettronica viene consegnata localmente, senza attivare una connessione SMTP. In pratica, consente di definire anche i domini virtuali che fanno capo allo stesso nodo locale. Come si vede dalla sintassi, si tratta di un elenco di stringhe, separato attraverso due punti verticali. Dovrebbe essere conveniente indicare sempre almeno i domini *localhost* e *localhost.localdomain*, nell'ipotesi che qualcuno usi indirizzi del tipo *tizio@localhost*, per quanto ciò possa sembrare strano assurdo.
`local_domains_include_host = {true|false}`	Attivando questa opzione ('**true**'), si fa in modo che il nome del nodo locale ottenuto dal sistema operativo, venga incluso automaticamente nell'elenco di domini locali ('**local_domains**').
`local_domains_include_host_literals = {true|false}`	Attivando questa opzione ('**true**'), si fa in modo di accettare messaggi, destinati al nodo locale, che fanno uso dell'indirizzo IP in forma letterale, al posto di un nome a dominio (secondo la forma ***nome*** *@[* ***indirizzo_ipv4*** *]*).
`forbid_domain_literals`	La presenza di questa opzione impedisce in generale l'uso di indirizzi numerici al posto del nome a dominio.
`log_level =` ***n_livello***	Permette di definire il livello di dettaglio per le informazioni memorizzate nei file delle registrazioni. Il valore zero corrisponde al minimo; valori superiori aumentano le informazioni (sei dovrebbe essere il valore che genera la massima quantità di notizie). Se questa opzione non viene dichiarata, il livello predefinito è cinque.
`message_size_limit =` ***dimensione***	Permette di fissare un tetto massimo alla dimensione dei messaggi in transito. Qui si usano normalmente i moltiplicatori '**K**' o '**M**'.

Directory	Descrizione
`never_users = `*utente*`[ : `*utente*`]`...	Permette di escludere il recapito di messaggi a utenti determinati, a meno che sia stato specificato un alias adatto nel file '`/etc/aliases`'. In pratica, serve per indicare l'elenco degli utenti di sistema, a cominciare da '**`root`**', che non possono o non dovrebbero ricevere posta.
`host_accept_relay = `*nodo*`[ : `*nodo*`]`... `host_accept_relay = ↩` `↪`*indirizzo_ipv4*`[ / `*n_bit_maschera*`][ : ... ]`...	Permette di definire a quali nodi è consentito usare il servente locale in qualità di relè. Questi nodi possono essere specificati per nome, per indirizzo IP o per gruppi di indirizzi IP.
`relay_domains = `*nome_di_dominio*`↩` `↪[ : `*nome_di_dominio*`]`...	Permette di elencare i domini per i quali si consente al servente di funzionare come relè.
`relay_domains_include_local_mx = ↩` `↪{true|false}`	Attivando l'opzione, si fa in modo di consentire la funzionalità di relè verso i domini che, secondo il DNS, dovrebbero essere serviti dal nodo locale. In pratica, si fa in modo di seguire la configurazione definita attraverso il DNS, con i record '**`MX`**', con cui si stabilisce che tale servente SMTP si deve occupare della consegna presso quei domini, accettando messaggi provenienti da qualunque dominio esterno.
`spool_directory = `*directory*	Definisce il percorso della directory usata come coda dei messaggi da Exim. Generalmente potrebbe trattarsi di '`/var/spool/exim/`', che poi si articola ulteriormente.
`trusted_users = ↩` `↪`*utente_fidato*`[ : `*utente_fidato*`]`... `trusted_groups = ↩` `↪`*gruppo_fidato*`[ : `*gruppo_fidato*`]`...	Definisce quali processi possono passare messaggi a Exim, specificando il mittente attraverso l'opzione '**`-f`**' della riga di comando. Tali processi sono accettati in quanto avviati con i privilegi dell'utente o del gruppo indicati. Se non viene specificata alcuna di queste due opzioni, ciò può essere fatto solo da un processo con i privilegi dell'utente '**`root`**'. Solitamente si definisce in questo modo l'utente '**`exim`**' o l'utente '**`mail`**' (senza indicare alcun gruppo); potrebbe essere conveniente aggiungere altri utenti nel caso si vogliano gestire delle liste (*mailing-list*) con caratteristiche particolari.

Configurazione dei driver

La seconda, la terza e la quarta parte del file di configurazione sono dedicate alla definizione delle istanze dei driver di trasporto, di direzione (*director*) e di instradamento.

In queste parti, le direttive del file di configurazione sono suddivise a gruppetti, ognuno riferito alla definizione di un'istanza particolare. In pratica, appare la dichiarazione del nome dell'istanza che termina con due punti verticali, seguita da una riga contenente la dichiarazione del driver di riferimento, oltre a una serie di altre righe opzionali contenenti le impostazioni che gli si devono applicare (quando quelle predefinite non vanno bene).

nome_di_istanza_del_driver :
    `driver = `*nome_del_driver*
    [*direttiva_di_opzione*]
    [*direttiva_di_opzione*]
    ...

Le opzioni che possono essere indicate, si distinguono in generiche e private. Le opzioni generiche possono essere utilizzate con tutti i driver di uno stesso tipo (trasporto, direzione, instradamento), mentre quelle private si riferiscono solo a driver particolari. La direttiva che definisce il driver è un'opzione generica, che deve essere posta all'inizio, come mostra lo schema sintattico.

Nelle prime versioni di Exim è necessario separare le opzioni con una virgola, mettendo prima le opzioni generiche e dopo quelle specifiche; inoltre, il passaggio da opzioni generiche a opzioni specifiche dove essere segnalato con un punto e virgola. Nelle versioni più recenti queste restrizioni non esistono e non è richiesta l'indicazione di virgole o punti e virgola. Tuttavia, l'informazione viene riportata a spiegazione del motivo per il quale diversi esempi di configurazione in circolazione hanno queste virgole e questi punti e virgola, qua e là, senza un motivo apparente.

A titolo di esempio vengono mostrate e descritte un paio di dichiarazioni significative, che appaiono anche nell'esempio completo mostrato più avanti.

```
local_delivery:
  driver = appendfile
  file = /var/mail/${local_part}
```

Nella configurazione del trasporto, definisce l'istanza '**`local_delivery`**' del driver '**`appendfile`**'. Dal nome si intende che si tratta del trasporto che si deve occupare di consegnare localmente la posta elettronica.

Attraverso il driver '**`appendfile`**' si ottiene di aggiungere i messaggi a un file già esistente, specificato attraverso l'opzione '**`file`**': in questo caso si tratta di '**`/var/mail/${local_part}`**', che in pratica si espande in un file denominato come l'utente che deve riceverlo, collocato nella directory '`/var/mail/`'.[2]

```
localuser:
  driver = localuser
  transport = local_delivery
```

Questo esempio fa riferimento alla configurazione del sistema di direzione; il nome dell'istanza è lo stesso di quello del driver, ma si tratta di cose differenti. Si può osservare la dichiarazione del trasporto utilizzato: '**`local_delivery`**', cioè il tipo di trasporto (l'istanza) già vista nell'esempio precedente.

Configurazione dei tentativi ripetuti

La penultima parte del file di configurazione, serve a definire il modo in cui scandire la ripetizione dei tentativi di invio (o di consegna) della posta. Ciò permette di distinguere il comportamento in base al dominio di destinazione e al tipo di errore che ha impedito la consegna del messaggio. Generalmente si trova già un esempio generico sufficiente.

Gli intervalli con cui vengono ripetuti i tentativi, devono tenere conto della frequenza con cui viene riavviato il processo di scansione della coda. Per esempio, se viene avviato '**exim**' con l'opzione '**-q30m**', che, come viene descritto in seguito, richiede il controllo della coda ogni 30 minuti, è poco sensato specificare nella configurazione intervalli inferiori, perché non potrebbero essere rispettati.

Configurazione della riscrittura degli indirizzi

L'ultima parte della configurazione è generalmente assente, o senza direttive. Serve a definire delle regole di alterazione sistematica degli indirizzi.

Per comprendere il problema viene descritto un caso pratico che potrebbe interessare. Quando si passa da Sendmail a Exim, potrebbe sentirsi la necessità di fare in modo che gli indirizzi ***nome*+*qualcosa*@*dominio***, vengano consegnati a ***nome*@*dominio***. Alcuni utenti potrebbero utilizzare questo trucco (comune per Sendmail) per distinguere la fonte da cui lo scrivente può avere tratto il loro indirizzo e avere implicitamente un'idea del contesto per il quale viene inviato ogni messaggio.

Exim ha dei meccanismi più potenti, ma quando si passa da Sendmail a Exim, gli utenti potrebbero desiderare di mantenere le vecchie convenzioni. La direttiva seguente dovrebbe risolvere il problema.

```
^(..*?)\+(.*)@(..*)$ $1@$3 T
```

Come si vede, attraverso un'espressione regolare vengono estratti gli elementi che contano dall'indirizzo, che poi viene ricostruito senza la parte superflua che ne impedirebbe il recapito.

Un esempio più semplice di questo problema è quello di una rete locale che utilizza nomi a dominio inesistenti nella rete esterna, pertanto si vuole sostituire l'indicazione dei domini locali con un dominio che esiste realmente:

```
*@brot.dg                $local_part@linuxdidattica.org E
*@localhost              $local_part@linuxdidattica.org E
*@localhost.localdomain  $local_part@linuxdidattica.org E
```

In questo modo, tutti i campi della busta del messaggio che corrispondono ai modelli indicati, vengono rimpiazzati con un indirizzo del dominio *linuxdidattica.org*.

Configurazione generica

Le versioni più recenti di Exim vengono distribuite normalmente con una configurazione generica, ben commentata, sufficiente al recapito locale dei messaggi, senza la possibilità di ricevere messaggi dall'esterno o di inviarne dall'interno.

L'attenzione maggiore va posta naturalmente a tutte quelle direttive che incorporano la parola chiave '**relay**', come '**relay_domains**' per esempio; inoltre diventa indispensabile verificare che la posta locale sia recapitata esattamente dove previsto, con la direttiva '**local_delivery**'.

Interazione con Procmail

Procmail è un programma per l'elaborazione dei messaggi di posta elettronica, che può essere usato per vari fini. La configurazione predefinita di Exim prevede spesso la presenza di direttive per passare a Procmail il compito di recapitare i messaggi locali, come nell'esempio seguente:

```
#
# Transports configuration.
#
…
#
# This transport is used for procmail.
#
procmail_pipe:
  driver = pipe
  command = "/usr/bin/procmail"
  return_path_add
  delivery_date_add
  envelope_to_add
  suffix = ""
```



```
…
#
# Directors configuration
#
…
#
# This director runs procmail for users who have a .procmailrc file
#
procmail:
  driver = localuser
  transport = procmail_pipe
  require_files = ${local_part}:+${home}:+${home}/.procmailrc:+/usr/bin/procmail
  no_verify
```

Senza entrare nel dettaglio del significato delle direttive, basta osservare che la presenza del file '`~/.procmailrc`', assieme al programma ('`/usr/bin/procmail`'), fa sì che questo tipo di recapito venga attivato. Di conseguenza, con tale situazione, **la configurazione errata del file '`~/.procmailrc`' potrebbe causare un funzionamento errato nel recapito dei messaggi locali**.

Avvio di Exim

L'avvio di Exim, allo scopo di attivare il servizio SMTP, avviene di solito attraverso la procedura di inizializzazione del sistema, come processo indipendente dal supervisore dei servizi di rete, anche se questa ultima possibilità è comunque consentita. Ma Exim può essere avviato anche per altri motivi, in particolare per ricevere un messaggio dallo standard input, da recapitare in qualche modo, oppure per ripassare i messaggi rimasti in coda, per ritentare il loro invio.

A seconda dello scopo per il quale viene avviato l'eseguibile '**exim**', possono essere richiesti dei privilegi particolari. Per la precisione, si distingue tra utenti comuni e amministratori. L'amministratore è l'utente '**root**', l'utente abbinato a Exim (normalmente '**exim**') e gli utenti definiti attraverso l'opzione '**trusted_users**'.

Uno dei motivi per cui può essere più conveniente avviare il servizio SMTP di Exim, in modo indipendente dal supervisore dei servizi di rete, è il fatto di poter affidare al demone Exim, così avviato, anche il compito di provvedere alla gestione dei messaggi in coda in modo automatico. Se si utilizza il controllo del supervisore dei servizi di rete, occorre affidare il lavoro di gestione della coda a un altro processo.

Quando si usa Exim come demone, cioè in modo autonomo dal supervisore dei servizi di rete, si usa l'opzione '**-bd**', seguita quasi sempre da '**-q*tempo***', che serve a specificare l'intervallo di scansione della coda di messaggi in attesa.

Se si vuole gestire il servizio SMTP attraverso il controllo del supervisore dei servizi di rete, occorre specificare l'opzione '**-bs**' e si deve dichiarare una riga simile a quella seguente nel file '`/etc/inetd.conf`'.

```
smtp    stream  tcp     nowait  root    /usr/sbin/exim  exim -bs
```

La riga di comando dell'eseguibile '**exim**' si può rappresentare sinteticamente così:

```
exim [opzioni]
```

Di seguito vengono elencate solo alcune opzioni assolutamente indispensabili, che servono a rendere l'idea delle funzioni di questo eseguibile.

Directory	Descrizione
`-bd`	Avvia '**exim**' come demone in attesa di connessioni SMTP. '**exim**' può essere avviato in questo modo solo da un amministratore; comunque ciò avviene di solito per mezzo della procedura di inizializzazione del sistema. Quando '**exim**' viene avviato con questa opzione, ma in modo manuale, può essere conveniente aggiungere l'uso delle opzioni diagnostiche '**-d**' o '**-dm**'.

Directory	Descrizione
`-bs`	Avvia '**`exim`**' in modo che questo si metta in attesa di ricevere messaggi dallo standard input, rispondendo poi a questi attraverso lo standard output. Questa opzione viene usata normalmente per gestire l'avvio di '**`exim`**' attraverso il supervisore dei servizi di rete.
`-bp`	Questa opzione permette di visualizzare l'elenco dei messaggi rimasti in coda per qualche motivo. Il funzionamento non è perfettamente identico a Sendmail, in quanto solo un utente con i privilegi necessari può ottenere queste informazioni.
`-bt` [*indirizzo*]	Avvia '**`exim`**' in una modalità di verifica degli indirizzi. Se viene indicato un indirizzo come argomento dell'opzione, si ottengono le informazioni essenziali sulla consegna verso tale destinazione. Ciò permette di verificare la correttezza della configurazione, dal momento che si ottiene anche l'indicazione del tipo di direzione e di trasporto utilizzati. Se non viene specificato l'indirizzo nella riga di comando, '**`exim`**' funziona in modo interattivo, proponendo un invito (il simbolo '**`>`**') per l'inserimento di ogni indirizzo da verificare (per terminare si può utilizzare la combinazione [*Ctrl c*]).
`-d`[*n_livello*]	Permette di avviare '**`exim`**' in modo diagnostico, allo scopo di visualizzare informazioni attraverso lo standard error. Dopo la lettera dell'opzione, può essere aggiunto un numero che serve a rappresentare l'entità di informazioni desiderate: uno rappresenta un livello minimo, che viene utilizzato se non si specifica alcun numero, mentre un valore più grande rappresenta più informazioni.
`-dm`	Permette di ottenere informazioni diagnostiche riferite all'allocazione e alla deallocazione di memoria.
`-q`	L'opzione '**`-q`**' può avere un argomento, ma se usata da sola, fa in modo che '**`exim`**' esegua una scansione (una soltanto) dei messaggi in coda, tentando di consegnare ogni messaggio trovato al suo interno. La scansione non segue un ordine preciso e alla sua conclusione '**`exim`**' termina di funzionare. Questa opzione può essere usata solo da un amministratore.
`-q`*tempo*	L'opzione '**`-q`**', seguita dall'indicazione di una durata temporale, fa in modo che '**`exim`**' esegua una scansione della coda in modo ripetitivo, a intervalli della durata specificata dall'argomento. In questo modo, '**`exim`**' deve continuare a funzionare a tempo indeterminato. Questa opzione, con argomento, viene usata preferibilmente per l'avvio di '**`exim`**' come demone, in modo tale che possa prendersi cura sia del servizio SMTP che della verifica della coda. L'intervallo specificato in questo modo, determina in pratica il tempo minimo che può essere indicato nella configurazione dei tentativi ripetuti.
`-v`	È sinonimo di '**`-d1`**' (diagnosi di livello minimo).

Come accade spesso nei sistemi Unix, l'eseguibile '**`exim`**' può essere avviato utilizzando nomi diversi che definiscono implicitamente l'uso di opzioni determinate, che potrebbero essere difficili da ricordare. Non sempre i pacchetti di Exim includono tutti i collegamenti

che potrebbero essere utili. Vale quindi la pena di riassumere quelli più comuni, che potrebbero essere realizzati utilmente se mancano nel proprio pacchetto.

Directory	Descrizione
'`/usr/lib/sendmail`', '`/usr/sbin/sendmail`'	Questi due collegamenti sono praticamente indispensabili se si vogliono utilizzare gli MUA comuni, cioè i programmi come Mailx ('**`mail`**'), che per l'invio dei messaggi si avvalgono proprio dell'eseguibile '**`sendmail`**'. Exim riconosce molte delle opzioni di Sendmail e in tal modo è possibile usare tali collegamenti. Secondo la logica dei sistemi GNU, l'eseguibile '**`sendmail`**' dovrebbe trovarsi esclusivamente nella directory '`/usr/sbin/`', ma per rispettare la tradizione è meglio aggiungere anche l'altro ('`/usr/lib/sendmail`') perché si vedono ancora script che fanno affidamento su questo.
'`/usr/bin/mailq`'	Quando '**`exim`**' viene avviato con il nome '**`mailq`**', permette di conoscere lo stato della coda, come se fosse stato avviato con l'opzione '**`-bp`**'.
'`/usr/bin/runq`'	Quando '**`exim`**' viene avviato con il nome '**`runq`**', fa in modo che '**`exim`**' esegua una singola scansione della coda, cercando di inviare i messaggi rimasti in attesa. Ciò, in pratica, come se fosse stato avviato con l'opzione '**`-q`**'.

Code e registri

Molto probabilmente (dipende da come è stato configurato in fase di compilazione), la directory '`/var/spool/exim/`' si articola in varie sottodirectory destinate a contenere informazioni variabili di vario tipo, tra cui le code dei messaggi e i file delle registrazioni.

La directory '`input/`' contiene precisamente i file delle code. Per ogni singolo messaggio che venga messo in attesa, si formano almeno due file: uno che termina con la sigla '`-D`' (*data*), che contiene il corpo del messaggio, e uno che termina con la sigla '`-H`' (*head*), che contiene le altre informazioni. Se un messaggio è diretto a diversi destinatari, la sua consegna può richiedere molto tempo e l'annotazione delle destinazioni presso cui è stato recapitato con successo. In tal caso viene creato un terzo file, che termina con la sigla '`-J`' (*journal*), all'interno del quale si annotano gli indirizzi già raggiunti.

Se un messaggio finisce in coda, ci deve essere un motivo. Nella directory '`msglog/`' vengono annotati file con gli stessi nomi utilizzati per i dati in coda, senza sigle finali, contenenti l'elenco degli insuccessi accumulati durante i vari tentativi ripetuti.

Se si decide di intervenire in modo brutale nei file delle code, cancellandoli, ci si deve ricordare di eliminare anche i file corrispondenti della directory '`msglog/`'.

Naturalmente sono disponibili anche dei file di registrazioni veri e propri, che potrebbero trovarsi in '`/var/spool/exim/log/`', oppure, più convenientemente, in '`/var/log/exim/`'. Si tratta di tre file: '`mainlog`', '`rejectlog`', '`processlog`' e '`paniclog`'. Il significato di questi nomi dovrebbe essere intuitivo: '`mainlog`' è l'archivio principale delle operazioni compiute, in cui si segnalano l'arrivo e la consegna di ogni messaggio; '`rejectlog`' registra le informazioni sui messaggi il cui transito è rifiutato in funzione della configurazione; '`processlog`' serve a segnalare l'effetto della ricezione di alcuni segnali (come '**`SIGHUP`**' e '**`SIGUSR1`**'); infine, '`paniclog`' permette di annotare le situazioni di errore che Exim non riesce a gestire.

Attraverso l'opzione '**`log_level`**' del file di configurazione, è possibile definire il livello di dettaglio delle informazioni che appaiono nel file delle registrazioni. Il valore predefinito corrisponde comunque a un buon livello di dettaglio.

Con l'opzione '**`preserve_message_logs`**', attivandola, è possibile evitare la cancellazione dei file delle registrazioni collocati nella directory '`msglog/`'. Ciò può essere utile solo nel caso in cui si volesse fare un controllo approfondito degli errori che si verificano durante i vari tentativi di consegna.

Archiviazione dei file delle registrazioni

Le distribuzioni GNU dovrebbero essere organizzate per gestire in modo elegante l'archiviazione dei file delle registrazioni, spezzando i file in parti che contengono periodi relativamente brevi, solitamente distinte attraverso un'estensione numerica progressiva che indica l'età relativa del file.

Exim fornisce un proprio script per svolgere questo compito, '**`exicyclog`**', che può essere usato quando la propria distribuzione GNU non dovesse già provvedere per conto proprio.

Per avviarlo, si potrebbe mettere un'istruzione come quella seguente nel file '`/etc/crontab`' (ammesso che lo script si trovi nella directory '`/usr/sbin/`').

```
01 0 * * * root /usr/sbin/exicyclog
```

Riferimenti

- *Exim*
  http://www.exim.org/

[1] **Exim** GNU GPL

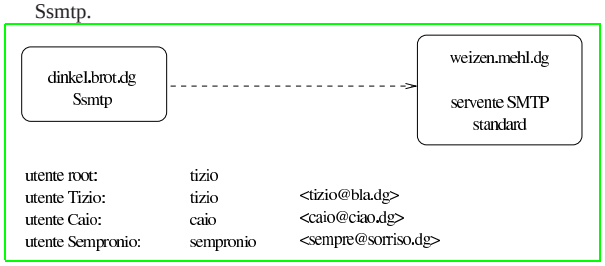
[2] La directory in questione deve avere i permessi 1777_8, altrimenti non può funzionare il sistema di blocco dei file (*lock*) e in pratica i messaggi non vengono recapitati.

Ssmtp

Configurazione 303
Ricezione della posta elettronica 304
Considerazioni sulla sicurezza 304

Ssmtp [1] è un sistema molto semplice per l'invio della posta elettronica a un servente SMTP. Lo scopo di questo programma è quello di sostituire l'MTA tipico, quando non è necessaria la gestione della consegna locale (MDA) e si può fare affidamento continuamente su un MTA esterno, contattato attraverso il protocollo SMTP.

Figura u53.1. Esempio di una situazione in cui viene utilizzato Ssmtp.



Per comprendere il funzionamento di Ssmtp è necessario partire da un esempio, come si vede nella figura u53.1. Nell'elaboratore *dinkel.brot.dg* è stato installato Ssmtp; l'elaboratore *weizen.mehl.dg* offre un servente SMTP a cui *dinkel.brot.dg* può accedere. Nell'elaboratore *dinkel.brot.dg* sono registrati tre utenti: Tizio, Caio e Sempronio, ognuno dei quali riceve la propria posta elettronica presso una casella remota (come indicato nella figura stessa); inoltre, è Tizio che svolge anche il ruolo di amministratore.

Ssmtp non è un demone e viene avviato solo quando serve. In quel momento, contatta il servente SMTP indicato nella sua configurazione e gli affida il messaggio. Pertanto, il servente in questione deve essere sempre disponibile, ovvero deve essere sempre disponibile la comunicazione verso quell'elaboratore remoto. Da questo si comprende che Ssmtp non può essere usato in un elaboratore che si collega alla rete esterna solo saltuariamente.

In generale, i messaggi destinati all'elaboratore locale non si possono consegnare, ma anche se non si intende usare un programma come Mailx ('**`mail`**') non bisogna dimenticare che in un sistema Unix standard vengono generati automaticamente dei messaggi che spesso sono diretti all'amministratore, per aggiornarlo sullo svolgimento di operazioni periodiche o per avvisarlo di qualunque tipo di problema. In altri termini, un sistema Unix non può fare a meno di un programma che consenta almeno l'invio dei messaggi di posta elettronica. Ssmtp è in grado di ridirigere questi messaggi a un indirizzo di posta elettronica, che però deve esterno all'elaboratore locale.

Quando un utente invia un messaggio, è necessario che Ssmtp sia in grado di modificare il campo '**`From:`**', in modo che appaia un indirizzo di posta elettronica valido; diversamente, se apparisse l'indirizzo dell'elaboratore locale (*dinkel.brot.dg* nell'esempio), non potrebbe ricevere alcuna risposta dal suo interlocutore.

Configurazione

Ssmtp utilizza solo due file per la configurazione. Si tratta di '`/etc/ssmtp/ssmtp.conf`' e di '`/etc/ssmtp/revaliases`'. È sufficiente mostrare degli esempi compatibili con quanto visto in figura u53.1 per comprendere l'uso delle direttive di questi file. Si comincia con '`/etc/ssmtp/ssmtp.conf`', dove i commenti relativi alle direttive non utilizzate sono rimasti in inglese:

```
# /etc/ssmtp/ssmtp.conf

# La persona che riceve i messaggi diretti agli utenti con UID inferiore a 10
root=tizio@bla.dg

# Il servente SMTP per l'invio dei messaggi.
mailhub=weizen.mehl.dg

# Where will the mail seem to come from?
#rewriteDomain=dinkel.brot.dg

# Il nome completo del nodo locale
hostname=dinkel.brot.dg

# Set this to never rewrite the "From:" line (unless not given) and to
# use that address in the "from line" of the envelope.
#FromLineOverride=YES
```

Come si vede, i messaggi diretti a *root@localhost* vengono modificati e inviati a *tizio@bla.dg*; l'invio dei messaggi avviene facendo uso del servizio offerto da *weizen.mehl.dg*; il nome dell'elaboratore locale è *dinkel.brot.dg*.

Quanto visto fino a questo punto basta per inviare i messaggi, ma non è sufficiente a modificare il campo '**`From:`**', perché si tratta di un compito affidato alla configurazione con il file '`/etc/ssmtp/revaliases`':

```
# /etc/ssmtp/revaliases

root:           tizio@bla.dg
tizio:          tizio@bla.dg
caio:           caio@ciao.dg
sempronio:      sempre@sorriso.dg
```

Si può osservare che al nominativo-utente si abbina l'indirizzo di posta elettronica appropriato, compreso il caso di '**`root`**'.

Ricezione della posta elettronica

La ricezione della posta elettronica è un'attività al di fuori della competenza di Ssmtp, pertanto la si deve prelevare attraverso il protocollo POP3 o un altro simile, ma questo direttamente attraverso il programma utilizzato per la lettura della stessa. In altri termini, non si può usare Fetchmail (39.11.3) e nemmeno altri programmi che rinviano i messaggi prelevati nel sistema di recapito locale.

Considerazioni sulla sicurezza

L'utilizzo di Ssmtp si giustifica solo quando si vuole evitare di dovere gestire un servente SMTP standard, con tutti i problemi di sicurezza che ciò comporta. Inoltre, si evita di avere in funzione continuamente il programma, riducendo così le risorse elaborative richieste.

[1] **Ssmtp** GNU GPL

Usenet

Introduzione a Usenet 307
Software per Usenet 307
Organizzazione generale del sistema di news di Usenet ... 308
Organizzazione del software per la gestione delle news ... 311
Riferimenti 312
Introduzione a INN -- InterNet News 313
Installazione e quadro generale di INN 313
Configurazione minima per l'uso locale puro e semplice .. 315
Demoni e altri programmi per l'uso minimo di INN 321
Feed in ingresso utilizzando il protocollo NNTP 323
Feed continuo in uscita utilizzando il protocollo NNTP ... 324
Feed periodico in uscita utilizzando il protocollo NNTP ... 326
Ritrasmissione di articoli attraverso la posta elettronica ... 327
Prelievo di articoli utilizzando il protocollo NNTP 327
Replicazione dei gruppi di un altro sito 328
Riferimenti 328

Introduzione a Usenet

Software per Usenet 307

Organizzazione generale del sistema di news di Usenet 308

- Selezione dei gruppi e feed 308
- Gestione degli articoli 309
- Protocollo NNTP 310
- Messaggi di controllo 310
- Distribuzioni 310

Organizzazione del software per la gestione delle news 311

- Collocazione locale degli articoli 311
- Gruppi amministrativi 311
- File amministrativi 311
- Distribuzione degli articoli 312
- Utente specifico 312

Riferimenti 312

Usenet è una sorta di rete astratta il cui scopo è quello di gestire l'organizzazione e la diffusione di un sistema pubblico di messaggi, ovvero di ***articoli***. Per «rete astratta» si intende qualcosa che va oltre i confini di una rete avente una sua tecnologia particolare: anche se oggi la rete più diffusa è Internet, Usenet non è necessariamente qualcosa che riguardi esclusivamente questo tipo di supporto.[12]

L'idea alla base della nascita di Usenet è stata quella di riuscire a realizzare un sistema automatico di diffusione di articoli tra un gruppo di elaboratori, in modo da permettere agli utenti di questi di leggerli e di poter aggiungere i propri. Inizialmente il meccanismo attuato era molto semplice: a intervalli regolari (magari solo una volta al giorno) ogni nodo impacchettava gli articoli di cui disponeva e li spediva ai suoi nodi corrispondenti, i quali provvedevano poi a selezionare quelli che non avevano già ricevuto da altri (eliminando così gli articoli doppi).

L'evoluzione di Usenet ha portato all'introduzione di tecniche di diffusione più dinamiche, soprattutto attraverso l'introduzione del protocollo NNTP all'interno di Internet. Tuttavia, la complessità della storia di questo sistema di messaggi ha portato ad altrettanta difficoltà nella configurazione e nell'amministrazione del software relativo.

Software per Usenet

Il primo software utilizzato per realizzare questo meccanismo di diffusione di articoli ha una sua storia importante: «A», «B» (Bnews), «C» (C News) e INN (InterNet News). All'interno di questa sequenza, nel 1986, ovvero subito prima che apparisse C News, si inserisce lo standard NNTP (*Network news transfer protocol*), il cui scopo è quello di dare un protocollo per il trasferimento degli articoli di Usenet all'interno di Internet (attraverso il TCP), ricalcando l'esperienza di SMTP (il protocollo per i messaggi di posta elettronica).

L'introduzione del protocollo NNTP, documentata dall'RFC 977, coincideva con la realizzazione del «demone NNTP», ovvero ciò che adesso è conosciuto come la «realizzazione di riferimento» (*reference implementation*). Questo demone si inserisce in pratica come un'interfaccia rispetto a sistemi di gestione delle news come Bnews e C News.

Da questa situazione un po' confusa emerge in particolare InterNet News, ovvero INN, che oltre a gestire il protocollo NNTP è in grado di amministrare anche lo strato sottostante che in altre circostanze sarebbe di competenza di Bnews o di C News.

Organizzazione generale del sistema di news di Usenet

Inizialmente si accennava al fatto che Usenet sia una sorta di rete sulle reti: in pratica è formata dall'insieme di nodi che offrono un servizio di pubblicazione e diffusione di articoli. Ogni nodo riceve da altri nodi gli articoli che gli interessano e probabilmente provvede a sua volta a fornirli ad altri aggiungendovi i propri. Questo meccanismo è il feed, con il quale si alimenta il flusso degli articoli di Usenet.

Le news di Usenet sono organizzate in ***gruppi*** (ovvero gruppi di discussione o *newsgroup*), all'interno dei quali gli utenti possono leggere gli articoli e spedirne dei nuovi (*post*). Questi gruppi di discussione sono denominati in modo gerarchico, utilizzando una notazione simile a quella dei nomi a dominio di Internet, senza avere però alcuna relazione con questi. L'idea che sta alla base della gerarchia dei gruppi di discussione di Usenet è la stessa delle directory in un file system. Per esempio, potrebbe esistere il gruppo '**`grano.farina`**', ma anche il gruppo '**`grano.farina.farro`**', così come '**`grano.sementi`**'; eventualmente potrebbe mancare il gruppo '**`grano`**' puro e semplice. Evidentemente, i nomi usati nella definizione della gerarchia danno già un'idea degli argomenti per i quali sono stati fatti.

Come si intuisce, la gerarchia si sviluppa da sinistra a destra, esattamente come nella notazione che si usa per rappresentare i percorsi all'interno di un file system, utilizzando però il punto per separarne i vari elementi.

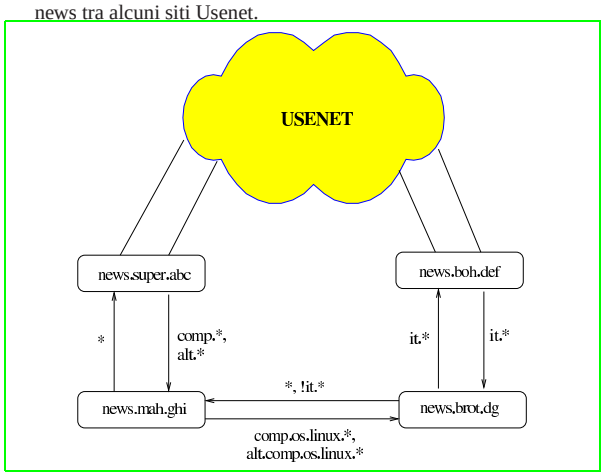
Abituati oggi con Internet, può risultare difficile la comprensione del meccanismo che sta alla base di Usenet. Ci si potrebbe domandare come mai non sia possibile contattare questo o quel gruppo di discussione raggiungendo semplicemente il nodo da cui ha origine. Il fatto è che non c'è un'origine vera e propria per un gruppo; tutto quanto è frutto di una cooperazione tra i vari siti che offrono accesso a Usenet e per raggiungere un gruppo occorre avere accesso presso uno di questi siti (uno che gestisca il gruppo a cui si è interessati).

La quantità e la varietà dei gruppi esistenti è tale per cui è diventato impossibile gestire tutti i gruppi esistenti in un solo «sito Usenet», soprattutto a causa del volume di traffico che si genererebbe nella rete. In questo senso, l'amministrazione di un servizio del genere comporta due problemi fondamentali: decidere i gruppi che si vogliono gestire e trovare i siti Usenet con cui comunicare per potervi partecipare.

Selezione dei gruppi e feed

Il feed è il meccanismo alla base del funzionamento di Usenet. In pratica è ciò che permette la diffusione degli articoli dei vari gruppi di discussione. Volendo predisporre un servente di news, ovvero un sito Usenet, occorre ottenere il feed dei gruppi a cui si è interessati, filtrando ciò che non interessa.

Figura u54.1. Esempio di come potrebbe funzionare il flusso di news tra alcuni siti Usenet.



Nell'esempio di figura u54.1 vengono mostrati dei nodi con nomi a dominio tipici di Internet, i quali collaborano al sistema di Usenet. In particolare, '**`news.super.abc`**' è inteso come un sito Usenet che dispone di una grande quantità di gruppi; da questo '**`news.mah.ghi`**' attinge i gruppi '**`comp.*`**' e '**`alt.*`**'. Inoltre, da un'altra parte dell'universo di Usenet appare '**`news.boh.def`**' che tra gli altri dispone dei gruppi '**`it.*`**'; questi vengono forniti in particolare a *news.brot.dg*. Ma a *news.brot.dg* non basta perché vuole avere anche i gruppi '**`comp.os.linux.*`**' e '**`alt.comp.os.linux.*`**': questi li ottiene da '**`news.mah.ghi`**'.

Tuttavia, non è sufficiente ottenere la copia degli articoli di questo o quel gruppo, bisogna considerare che ogni sito Usenet può aggiungere i propri e questi dovrebbero diffondersi su tutti gli altri siti Usenet che gestiscono il gruppo corrispondente; inoltre, ogni nodo potrebbe aggiungere i propri gruppi locali, che potrebbero o meno essere accolti anche da altri.[3]

Il feed che si vede indicato nella figura ha due direzioni: una per i gruppi da ricevere da un nodo e l'altra per i gruppi da inviare allo stesso. Per esempio, '**`news.mah.ghi`**' riceve solo i gruppi '**`comp.*`**' e '**`alt.*`**' da '**`news.super.abc`**', ma gli restituisce indietro tutto; precisamente, restituisce tutto quello che '**`news.super.abc`**' è disposto ad accettare. In questo modo, gli articoli spediti per mezzo di '**`news.mah.ghi`**' prendono la loro strada attraverso Usenet. Invece, il caso di *news.brot.dg* è più complesso, perché da una parte ha un feed che gli permette di accedere ai gruppi '**`it.*`**', dall'altra ne ha un altro per alcuni gruppi '**`comp`**' e '**`alt`**'. Dallo schema della figura si intende che questo mantenga un feed con '**`news.boh.def`**' esclusivamente per i gruppi '**`it.*`**'; dall'altra parte, nel collegamento con '**`news.mah.ghi`**' possono essere mandati tutti gli articoli che semplicemente non appartengono ai gruppi '**`it.*`**', poi è compito di '**`news.mah.ghi`**' escludere quello che non gli riguarda.

Gestione degli articoli

Data l'organizzazione di Usenet in cui gli articoli si distribuiscono attraverso percorsi non prevedibili, è necessario che ogni sito sia in grado di gestire intelligentemente ciò che lo riguarda. Per esempio occorre evitare di accettare articoli che sono già stati ricevuti in qualche modo; gestendo una quantità elevata di gruppi occorre dare una scadenza alla loro conservazione, eliminandoli periodicamente; inoltre è opportuno evitare di diffondere articoli attraverso nodi che sono già stati attraversati da questi.

Per ottenere questo risultato, il software che si utilizza per gestire le news deve avere un sistema di registrazione del traffico che lo riguarda, conservando anche le informazioni sugli articoli scaduti che nel frattempo sono stati cancellati localmente. Per distinguere gli articoli occorre un modo preciso e «sicuro»; questo si ottiene attraverso una stringa di identificazione, generata in qualche modo da

chi riceve per la prima volta l'articolo e inserita nell'intestazione del messaggio attraverso il campo '**Message-ID:**', come nell'esempio seguente:

```
Message-ID: <36F130C4.2E242945@brot.dg>
```

La scadenza di un articolo può essere indicata anche da chi lo scrive, attraverso il campo '**Expires:**'; il sistema di gestione locale delle news può stabilire una scadenza predefinita e anche una scadenza massima, senza rispettare necessariamente quanto richiesto dall'articolo stesso.

Ogni messaggio porta con sé anche l'informazione del percorso dei vari nodi di Usenet che ha attraversato nel campo '**Path:**'. La forma è quella del cosiddetto *bang path*, perché si tratta di una stringa in cui si utilizza il punto esclamativo per separare i vari nomi. Per esempio,

```
Path: roggen.brot.dg!dinkel.brot.dg
```

rappresenta il transito da *dinkel.brot.dg* a *roggen.brot.dg*. Evidentemente, conoscendo i suoi passaggi, si deve evitare che questo articolo sia ritrasmesso ai nodi che l'hanno già visto transitare. Quando ciò accade, quello è il punto in cui quella copia particolare dell'articolo si ferma.

Come nella posta elettronica, un articolo può essere diretto a più di un gruppo. In tal caso, i sistemi di gestione delle news che gestiscono tutti o alcuni di questi gruppi, dovrebbero riprodurre una sola copia «fisica» dell'articolo, mantenendo dei riferimenti all'interno di tutti i gruppi in cui questo è diretto. In pratica, nei sistemi Unix questo si ottiene con la tecnica dei collegamenti (fisici o simbolici che siano).

Protocollo NNTP

Prima di inserirsi in Internet, Usenet non aveva alcun bisogno del protocollo NNTP, ma adesso, con questo si semplificano molte cose. Di solito, un servente NNTP è anche un servente di news, dal quale un programma cliente, come può esserlo un navigatore grafico comune, può prelevare gli articoli e spedirne dei nuovi. Tuttavia, il protocollo NNTP fornisce anche un modo in più per ottenere il feed tra i vari siti di Usenet.

Messaggi di controllo

Più o meno come accade con le liste attraverso la posta elettronica, è possibile controllare in qualche modo il servizio delle news per mezzo di messaggi di controllo, contenenti campi particolari nell'intestazione. In questo caso si tratta del campo '**Control:**' e il comando più importante è '**cancel**', che dovrebbe permettere all'utente che ha spedito inizialmente un messaggio di ordinarne la cancellazione all'interno di tutta la rete Usenet. L'esempio seguente mostra in che modo potrebbe essere indicato questo comando:

```
Control: cancel <36F3CE29.F6176F5D@brot.dg>
```

Di solito un messaggio di questo tipo viene generato automaticamente dal programma utilizzato per accedere al servizio delle news, richiamando un comando particolare in base all'organizzazione del programma stesso. Tuttavia, è importante che il servente che lo riceve verifichi che si tratti verosimilmente dell'utente che aveva spedito originalmente l'articolo che adesso si richiede di cancellare.

La possibilità o meno di utilizzare i messaggi di controllo è regolata attraverso il file '`/etc/news/control.ctl`', che è documentato nella pagina di manuale *control.ctl(5)*.

Distribuzioni

Gli articoli possono distinguersi, oltre che per gruppi, anche per ***distribuzioni***. Si tratta del campo '**Distribution:**' che potrebbe apparire nell'intestazione di un messaggio. I nomi da attribuire alle distribuzioni possono servire a qualificare in modo alternativo gli articoli, per qualche scopo, per esempio per limitare la loro diffusione a un ambito locale o «regionale». Di sicuro si conoscono due distribuzioni comuni: '**world**' e '**local**'. In linea di massima si può dire che un articolo fatto per la distribuzione '**world**' dovrebbe essere lasciato diffondersi su tutti i siti Usenet, mentre un altro articolo etichettato per la distribuzione '**local**', dovrebbe essere inteso per un uso locale riferito al sito Usenet attraverso cui è stato spedito.

```
Distribution: local
```

L'esempio mostra in che modo potrebbe essere composto il campo '**Distribution:**' quando viene utilizzato il nome '**local**'.

Di solito, l'utilizzatore normale non si cura di questo campo nell'intestazione dei suoi articoli e probabilmente non ha nemmeno il modo di aggiungerlo. In questo senso è poi il software utilizzato per la gestione delle news che dovrebbe essere configurato in modo da attribuire un valore predefinito, eventualmente in base al gruppo in cui è stato spedito.

Organizzazione del software per la gestione delle news

Come già accennato, il software per la gestione di un sito Usenet segue un filone più o meno continuo, stabilendo implicitamente il legame tra Usenet e Unix. Il software più comune in questo momento è composto da C News, a cui si abbina normalmente un servizio NNTP, e da INN, che al contrario di C News può fare tutto da solo. Tra questi ci sono alcune affinità importanti la cui conoscenza può facilitare lo studio di ciò che si intende utilizzare effettivamente.

Collocazione locale degli articoli

Su un sistema GNU, gli articoli di un servizio di news sono collocati generalmente a partire dalla directory '`/var/spool/news/`', con una struttura che ricalca quella del nome dei gruppi di discussione. Per esempio, il gruppo '**comp.os.linux**' dovrebbe trovarsi nella directory '`/var/spool/news/comp/os/linux/`'. All'interno di ogni directory riferita a un gruppo particolare vengono collocati gli articoli in forma di file, denominandoli in modo numerico: '`1`', '`2`',... Quel numero serve solo come riferimento, in base a quanto organizzato dal servizio locale di gestione delle news.

Quando si riceve un messaggio destinato a più gruppi, dei quali tutti o alcuni di questi sono gestiti localmente, quello che si ottiene è la riproduzione di questi attraverso dei collegamenti (generalmente dei collegamenti fisici).

Gruppi amministrativi

Il software di gestione delle news richiede normalmente la presenza di alcuni gruppi locali per uso amministrativo, che non vanno eliminati. Potrebbe trattarsi di:

'**control**'	utilizzato per conservare i messaggi di controllo, per esempio gli ordini di rimozione degli articoli;
'**junk**'	utilizzato per raccogliere gli articoli dei gruppi che localmente risultano mancanti;
'**test**'	utilizzato a livello diagnostico per poter sperimentare l'invio di articoli;
'**to**'	utilizzato per scopi vari dal software con cui si gestisce il servizio.

File amministrativi

I file amministrativi, come per esempio lo storico degli articoli già visti, dovrebbero trovarsi nella directory '`/var/lib/news/`'. In particolare, in questa directory si dovrebbe trovare il file '`active`', contenente le informazioni necessarie a determinare quali siano i gruppi gestiti e i relativi intervalli di «numeri», ovvero dei nomi dei file rispettivi. In generale, dovrebbe essere sufficiente modificare questo file per creare o definire la gestione di un nuovo gruppo di discussione, tanto che di solito per avviare un servizio del genere si comincia prelevando il file '`active`' di un altro nodo Usenet e lo si modifica successivamente.

Distribuzione degli articoli

Il sistema di gestione delle news riceve in qualche modo gli articoli provenienti dai nodi corrispondenti, filtrando presumibilmente solo una parte dei gruppi e scartando i doppioni (ovvero ciò che in base allo storico risulta essere già stato visto localmente). Successivamente si deve occupare di dirigere una copia di questi articoli nel deposito locale, in modo da metterli a disposizione per la lettura a chi può averne accesso; inoltre si deve curare di far proseguire (in qualche modo) una copia di questi articoli ai nodi corrispondenti che non appaiano già nel percorso annotato nel campo '**`Path:`**'.

Nello stesso modo, si potrebbe definire l'accumulo di una copia degli articoli in un file di archivio, oppure anche verso un sistema di posta elettronica (probabilmente un indirizzo riferito a una lista).

Utente specifico

Generalmente, alla gestione del software per l'amministrazione delle news viene abbinato un utente di sistema, denominato '**`news`**', al quale viene aggiunto normalmente anche il gruppo '**`news`**'. La directory personale di questo utente fittizio dovrebbe essere '`/var/spool/news/`'. Tutti i file amministrativi, compresi quelli di configurazione che si trovano sotto '`/etc/news/`', assieme a tutto ciò che si trova a partire da '`/usr/lib/news/`', dovrebbe appartenere all'utente (e al gruppo) '**`news`**'.

Tutte le volte che l'amministratore del sistema deve intervenire sulla gestione del software delle news dovrebbe avere l'accortezza di utilizzare l'identità e i privilegi dell'utente '**`news`**', possibilmente attraverso il comando '**`su`**', oppure deve fare attenzione a sistemare la proprietà dei file che crea.

Quando si utilizza il sistema Cron per eseguire delle elaborazioni periodiche, occorre preoccuparsi di questo fatto; eventualmente si può utilizzare il file crontab dell'utente '**`news`**', oppure quello dell'utente '**`root`**', badando però a sistemare l'identità dell'utente:

```
su - news -c "/usr/lib/news/bin/news.daily"
```

Riferimenti

- Marco d'Itri, *Usenet*
  http://www.linux.it/~md/usenet/
- Brendan P. Kehoe, *Zen and The Art of the Internet*, 1992
  http://www.cs.indiana.edu/docproject/zen/zen-1.0_toc.html
- Brian Kantor, Phil Lapsley, *RFC 977, Network News Transfer Protocol*, 1986
  http://www.ietf.org/rfc/rfc977.txt
- Olaf Kirch, *NAG, The Linux Network Administrators' Guide*

[1] In altri termini, Usenet è stata il sistema BBS di Unix, prima attraverso il trasporto UUCP e poi estendendosi anche su Internet (oltre che su altri sistemi operativi).

[2] Il modo originale di scrivere il nome di questo sistema di messaggistica è USENET, utilizzando quindi solo le lettere maiuscole, ma successivamente si è diffusa la forma usata in questo documento, cioè con la sola iniziale maiuscola.

[3] L'aggiunta di un gruppo di discussione al di fuori dell'ambito locale è una cosa che deve essere fatta di comune accordo, secondo una procedura stabilita.

Introduzione a INN -- InterNet News

Installazione e quadro generale di INN 313
- Le variabili di ambiente 314
- Caratteri jolly 314

Configurazione minima per l'uso locale puro e semplice 315
- File «/etc/news/inn.conf» 315
- File «/etc/news/distrib.paths» 316
- File «/etc/news/newsfeeds» 317
- File «/etc/news/expire.ctl» 318
- File «/etc/news/nnrp.access» 319
- File «/var/lib/news/active» 320
- File «/var/lib/news/history» 320

Demoni e altri programmi per l'uso minimo di INN 321
- Avvio e conclusione del servizio NNTP 321
- Utilizzo di «ctlinnd» 322
- Operazioni di routine 323

Feed in ingresso utilizzando il protocollo NNTP 323
- File «/etc/news/hosts.nntp» 324
- File «/etc/news/incoming.conf» 324

Feed continuo in uscita utilizzando il protocollo NNTP 324
- File «/etc/news/innfeed.conf» 325
- File «/etc/news/newsfeeds» 325

Feed periodico in uscita utilizzando il protocollo NNTP 326
- File «/etc/news/nntpsend.ctl» 326
- File «/etc/news/newsfeeds» 326
- Utilizzo di «nntpsend» 326

Ritrasmissione di articoli attraverso la posta elettronica 327

Prelievo di articoli utilizzando il protocollo NNTP 327
- Utilizzo di «nntpget» 327

Replicazione dei gruppi di un altro sito 328

Riferimenti 328

INN, o InterNet News, è probabilmente il sistema più completo per la gestione delle news di Usenet. Purtroppo la sua documentazione non è soddisfacente, nel senso che presume una buona conoscenza di Usenet e il funzionamento del software precedente a INN (Bnews e C News). In questo capitolo viene dato un minimo di informazioni necessario per poter allestire un servizio di news chiuso, con qualche accenno alle possibilità di diffusione degli articoli assieme ad altri nodi, utilizzando il protocollo NNTP.

Installazione e quadro generale di INN

Per installare INN, data la sua complessità, è meglio se si dispone di un pacchetto già compilato e preparato per la propria distribuzione GNU. L'installazione dovrebbe utilizzare, in particolare, le collocazioni seguenti:

'`/var/spool/news/`'	l'inizio della gerarchia contenente gli articoli dei gruppi gestiti;
'`/var/lib/news/`'	l'inizio della gerarchia contenente i file amministrativi;
'`/usr/lib/news/`'	l'inizio della gerarchia contenente i programmi, gli script e le librerie;
'`/etc/news/`'	i file di configurazione che può essere opportuno modificare.

Naturalmente dovrebbero essere disponibili anche dei file per le pagine di manuale, collocati nelle posizioni consuete; inoltre dovrebbe

essere disponibile un minimo di documentazione assieme ad alcuni esempi di configurazione.

I programmi, cioè i file eseguibili e gli script, potrebbero trovarsi solo nella directory '`/usr/lib/news/bin/`', aggiungendo qualche collegamento nelle directory normali ('`/usr/bin/`' e '`/usr/sbin/`') solo per ciò che è stato ritenuto più importante.

Il funzionamento di INN dipende anche dall'esecuzione periodica di alcune operazioni amministrative, attraverso il controllo del sistema Cron. Per questo, se la propria distribuzione è stata organizzata anche in questo senso, è probabile che siano stati collocati alcuni script all'interno delle directory '`/etc/cron*/`', che poi vengono avviati in modo automatico in base alla configurazione predefinita di '`/etc/crontab`'.

Infine è da ricordare che tutti i file e le directory di INN devono appartenere all'utente (e probabilmente anche al gruppo) '**`news`**'. Anche i processi devono funzionare con i privilegi di questo utente amministrativo, con un'unica eccezione data dal programma '**`innd`**' che si occupa di fornire il servizio NNTP, che dovendo accedere alla porta '**`nntp`**', corrispondente al numero 119, deve avere inizialmente i privilegi dell'utente '**`root`**'.

Le variabili di ambiente

INN dipende da un reticolo di script e programmi che sono controllati da una serie di variabili di ambiente. Queste sono definite all'interno di pezzi di script che vengono incorporati dagli altri e che generalmente risiedono nella directory '`/usr/lib/news/lib/`'. Per la precisione, dal momento che gli script possono essere di vario tipo e si vuole lasciare la possibilità di estendere il sistema a piacere, esiste un file di dichiarazione di queste variabili per ogni interprete: '`innshellvars`' per la shell Bourne, '`innshellvars.csh`' per la shell C, '`innshellvars.pl`' per l'interprete Perl e '`innshellvars.tcl`' per l'interprete Tcl. Bisogna sapere che se si intende modificare qualcosa in uno di questi file (ma sarebbe meglio evitare di farlo), occorre ripetere le modifiche anche sugli altri.

Generalmente si vede l'utilizzo di '`innshellvars`', attraverso un'istruzione di incorporazione come quella seguente:

```
#!/bin/sh
…
. /usr/lib/news/lib/innshellvars
…
```

Caratteri jolly

In molte situazioni, i file di configurazione di INN ammettono l'uso di caratteri jolly (o metacaratteri), secondo le convenzioni stabilite nella pagina di manuale *wildmat(3)*. In linea di massima si può dire che si utilizzano le convenzioni normali riferite alle shell per l'uso dell'asterisco e del punto interrogativo. In particolare è ammesso anche la descrizione di intervalli di caratteri attraverso la notazione '**`[`**…**`]`**' e '**`[^`**…**`]`**'; inoltre è possibile togliere il significato speciale di un simbolo prefissandolo con una barra obliqua inversa.

Tabella u55.3. Modelli secondo *wildmat(3)*.

Modello	Descrizione
`\`***x***	Corrisponde al valore letterale di ***x***.
`?`	Un carattere singolo.
`*`	Qualunque sequenza di caratteri, anche nulla.
`[`***xy***…`]`	Un carattere singolo tra quelli indicati tra parentesi.
`[^`***xy***…`]`	Un carattere singolo esclusi quelli indicati tra parentesi.
`[`***x***`-`***y***`]`	Un carattere singolo tra l'intervallo di ***x*** e ***y***.
`[^`***x***`-`***y***`]`	Un carattere singolo escluso l'intervallo di ***x*** e ***y***.

Configurazione minima per l'uso locale puro e semplice

Il componente più importante di INN è il demone '**`innd`**'. Questo viene avviato tramite uno script che potrebbe essere necessario ritoccare a seconda del modo in cui si vuole organizzare il sistema. Potrebbe trattarsi di '`/etc/rc.d/rc.news`', o qualcosa di simile, ma per controllarlo dovrebbe essere stato predisposto un altro script, per esempio '`/etc/rc.d/init.d/innd`', in grado di accettare i soliti comandi ('**`start`**', '**`stop`**', ecc.) e che soprattutto lo avvii con i privilegi dell'utente '**`news`**'. Si osservi a questo proposito il commento introduttivo di '**`rc.news`**':

```
#!/bin/sh
##  $Revision: 1.19 $
##  News boot script.  Runs as "news" user.  Requires inndstart be
##  setuid root.  Run from rc.whatever as:
##     su news -c /path/to/rc.news >/dev/console
```

In una parte avanzata di questo script ci dovrebbe essere qualcosa che assomiglia al pezzo seguente, dove si intende che tutto dipende dal contenuto delle variabili di ambiente che si possono vedere:

```
##  Start the show.
echo 'Starting innd.'
eval ${WHAT} ${RFLAG} ${INNFLAGS}
```

I nomi e il numero delle variabili di ambiente indicate cambia da una distribuzione GNU all'altra, ma è importante sapere come viene avviato '**`innd`**' in modo preciso, perché a seconda di alcuni particolari della configurazione si devono utilizzare delle opzioni determinate. Analizzando il file che è stato usato per mostrare questo esempio, si osserva che:

```
##  Pick ${INND} or ${INNDSTART}
WHAT=${INNDSTART}
```

il comando per avviare '**`innd`**' proviene dal contenuto della variabile di ambiente '**`INNDSTART`**', che è stata definita all'interno di '`/usr/lib/news/lib/innshellvars`' (e dopo qualche ricerca si scopre che si tratta di '`/usr/lib/news/bin/inndstart`');

```
##  RFLAG is set below; set INNFLAGS in inn.conf(5)
RFLAG=""
```

Quindi si trova che la variabile '**`RFLAG`**' non contiene alcunché, ma potrebbe essere usata per inserire delle opzioni particolari; inoltre, da quanto si legge nel commento, la variabile '**`INNFLAGS`**' viene definita da qualche parte in base a una direttiva del file '`/etc/inn.conf`' e comunque dovrebbe essere inesistente.

In pratica, '**`innd`**' o '**`inndstart`**' dovrebbe essere avviato senza opzioni. Se ne vengono trovate, è bene toglierle, almeno fino a che non è stato superato il primo stadio di utilizzo di INN.

File «/etc/news/inn.conf»

Dal nome, '`/etc/inn.conf`', si intende che si tratti del file di configurazione più importante di INN. Il sistema di gestione dei pacchetti della propria distribuzione GNU/Linux dovrebbe provvedere a predisporlo in modo da permettere a INN di funzionare nel proprio sistema, ma è importante dargli un'occhiata ed eventualmente modificarlo. In generale conviene lasciare stare tutto com'è, tranne ciò che interessa.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all'interno della pagina di manuale *inn.conf(5)*. In breve, le righe vuote, quelle bianche e quelle che iniziano con il simbolo '`#`' vengono ignorate. Le direttive sono composte da una sorta di assegnamento descritto secondo la sintassi seguente:

nome **:** ***valore***

In particolare, tra i due punti che seguono il nome e il valore assegnato, ci deve essere almeno uno spazio orizzontale di qualunque tipo; inoltre, se il valore assegnato è rappresentato da una stringa contenente degli spazi, questa non deve essere racchiusa tra virgolette.

Le direttive su cui è molto importante intervenire sono poche; riguardano la definizione dell'«organizzazione» predefinita e i nomi

con cui si deve identificare il nodo che offre il servizio. L'organizzazione è un nome che viene abbinato al campo '**`Organization:`**' nell'intestazione degli articoli; di solito dovrebbe essere definito dal programma cliente dell'utente che spedisce l'articolo.

```
organization:           Azienda Brot
```

L'esempio mostra un'idea di ciò che potrebbe essere indicato come organizzazione. Si osservi il fatto che non sono state usate le virgolette per delimitare il nome. Questa informazione potrebbe essere definita anche attraverso la variabile di ambiente '**`ORGANIZATION`**', la quale, se esiste, prevale su quanto definito nel file '`/etc/inn.conf`'.

Un problema un po' più delicato riguarda invece la definizione delle direttive '**`server:`**', '**`fromhost:`**' e '**`pathhost:`**'. Per prima cosa conviene decidere il nome a dominio del servizio NNTP. Di solito si crea un alias opportuno, qualcosa che inizi per '**`news.*`**', come nell'esempio seguente:

```
server:                 news.brot.dg
```

Anche in questo caso c'è la possibilità di utilizzare una variabile di ambiente che, se esiste, prevale sulla direttiva: si tratta di '**`NNTPSERVER`**'.

Ma il nome canonico dell'elaboratore potrebbe essere *dinkel.brot.dg*. Nell'intestazione degli articoli deve apparire il campo '**`From:`**' e il campo '**`Path:`**'; per queste indicazioni si presentano due possibilità: il nome canonico dell'elaboratore oppure il nome a dominio utilizzato nella posta elettronica. In pratica, seguendo l'esempio si dovrebbero indicare le direttive seguenti:

```
pathhost:               dinkel.brot.dg
fromhost:               dinkel.brot.dg
```

Se però la rete locale identificabile con il nome *brot.dg* riceve la posta elettronica direttamente con il dominio *brot.dg* (*tizio@brot.dg*), potrebbe essere il caso di cambiare la definizione nel modo seguente:

```
pathhost:               brot.dg
fromhost:               brot.dg
```

File «/etc/news/distrib.paths»

Gli articoli possono distinguersi, oltre che per gruppi, anche per «distribuzioni». Si tratta del campo '**`Distribution:`**' che potrebbe apparire nell'intestazione di un messaggio. Se chi spedisce il messaggio non provvede a indicare il campo '**`Distribution:`**', è opportuno che sia stabilito qualche valore predefinito in base al gruppo a cui è diretto. Questo è lo scopo del file '`/etc/news/distrib.paths`'.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all'interno della pagina di manuale *distrib.paths(5)*. In breve, le righe vuote, quelle bianche e quelle che iniziano con il simbolo '`#`' vengono ignorate. Le direttive sono composte da record suddivisi in tre elementi separati da due punti verticali ('`:`'), secondo la sintassi seguente:

peso : ***modello*** : ***distribuzione***

Il primo elemento è un numero maggiore di zero che serve a stabilire un ordinare di importanza delle direttive quando un certo gruppo può corrispondere a più modelli di record differenti: in quel caso si prende quello con il peso maggiore. Si osservi che l'esistenza del peso in questi record, rende indifferente l'ordine in cui questi appaiono.

Il secondo elemento è il nome di un gruppo o un modello che utilizza caratteri jolly secondo la convenzione di *wildmat(3)*: quando un articolo che non ha il campo '**`Distribution:`**' nell'intestazione corrisponde a un modello (e non ce ne sono altri di peso maggiore che possono corrispondere) gli viene attribuita la distribuzione il cui nome si colloca nell'ultimo elemento di questo record.

```
1:*:world
10:test:local
10:test.*:local
10:local.*:local
```


L'esempio mostra una classificazione molto semplice: tutti gli articoli sono classificati come appartenenti alla distribuzione '**`world`**', tranne quelli del gruppo '**`test`**' e dei gruppi che iniziano per '**`test.`**' e '**`local.`**', che invece sono classificati come facenti parte della distribuzione '**`local`**'.

File «/etc/news/newsfeeds»

Il file '`/etc/news/newsfeeds`' è molto importante perché definisce in che modo è organizzato il feed, ovvero la propagazione dei gruppi. Inizialmente conviene occuparsi solo di ciò che si vuole ottenere attraverso la voce '**`ME`**', secondo la tradizione del software precedente a INN.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all'interno della pagina di manuale *newsfeeds(5)* e qui vengono descritti solo alcuni aspetti elementari. In breve, le righe vuote, quelle bianche e quelle che iniziano con il simbolo '`#`' vengono ignorate. Le direttive sono composte da record separati in elementi attraverso due punti verticali ('`:`') come descritto dalla sintassi seguente, potendo essere continuate su più righe quando alla fine di una riga appare il simbolo '`\`'.

nome_sito [**/** ***esclusione***] : ***modelli*** [**/** ***distribuzioni***] : ***opzioni*** : ***parametri***

Il primo elemento serve a definire il nome del sito verso cui si vuole siano inviati gli articoli. Si tratta di un nome relativo alla configurazione, in quanto il suo scopo potrebbe anche essere solo quello di creare un archivio degli articoli su un file. Spesso, per ciò che riguarda nomi riferiti a voci locali, si utilizza un punto esclamativo finale per evitare confusione con i nomi di siti reali. L'elemento può essere completato da un elenco di esclusione che si distingue in quanto separato da una barra obliqua ('**`/`**'). Ciò permette di indicare una serie di nomi di siti che, se presenti nel percorso dell'articolo (il campo '**`Path:`**'), fanno sì che questo venga escluso. Volendo essere più precisi, la sintassi per il primo elemento potrebbe essere espressa nel modo seguente:

nome_sito [**/** ***nome_escluso*** [**,** ***nome_escluso***] **...**]

Il secondo elemento serve a definire un elenco di modelli riferiti ai gruppi che si vogliono selezionare, seguito eventualmente da un elenco di distribuzioni. Se vengono indicate anche le distribuzioni, allora sono accettati solo gli articoli che appartengono a una di quelle. Volendo rendere con maggiore dettaglio la sintassi per il secondo elemento del record, si può definire lo schema seguente:

modello [**,** ***modello***] **...** [**/** ***distribuzione*** [**,** ***distribuzione***] **...**]

I modelli dei gruppi possono usare i soliti caratteri jolly e possono essere indicati anche in forma di negazione, attraverso il prefisso di un punto esclamativo. I nomi delle distribuzioni non possono contenere caratteri jolly, ma ammettono l'uso del punto esclamativo per negare un nome.

Gli ultimi due elementi del record sono un po' particolari e vengono descritti solo quando necessario.

Volendo realizzare un servizio locale e chiuso di news, all'interno di questo file è sufficiente collocare la direttiva seguente, eliminando o commentando tutto il resto.

```
ME:*::
```

Il nome '**`ME`**' è una parola chiave che rappresenta il sito locale; di conseguenza si tratta del record che definisce quali gruppi e quali articoli vengono gestiti o accettati. L'asterisco nel secondo elemento indica che sono accettati tutti i gruppi e non si fanno discriminazioni per quanto riguarda la distribuzione eventuale. Gli ultimi due elementi non servono per questo tipo di situazione e sono vuoti.

Volendo essere un po' più dettagliati, magari in previsione di un'apertura all'esterno, si potrebbero definire i modelli relativi ai grup-

pi che si pensa di gestire, comprendendo anche quelli che devono restare relegati all'ambito locale.

```
ME:*,!junk,!control*,!local*::
```

In questo caso si intendono ricevere tutti gli articoli di qualunque gruppo, escludendo il gruppo '**junk**', i gruppi che iniziano per '**control**' e quelli che iniziano per '**local**'. Questi divieti riguardano solo la possibilità di ricevere articoli da un sito Usenet corrispondente e non limitano invece l'invio locale.

```
ME:*,@alt.binaries.*,!junk,!control*,!local*::
```

Questa è un'estensione dell'esempio precedente, in cui si utilizza una notazione che non è ancora stata descritta: il simbolo '**@**' posto davanti al modello '**alt.binaries.***' stabilisce che non vengono accettati articoli che siano stati inviati anche ai gruppi di quel modello. In pratica, se un articolo è indirizzato a un gruppo della gerarchia '**alt.binaries**' e anche a gruppi che si intendono gestire, questo articolo viene escluso completamente.

In generale, se non si sanno gestire le distribuzioni, sarebbe meglio evitare di porre delle limitazioni a tale riguardo.

Inizialmente sarebbe bene limitarsi a gestire solo il record '**ME**', commentando tutto il resto se dovesse esserci qualcosa, verificando che il demone '**innd**' sia avviato senza argomenti particolari.

File «/etc/news/expire.ctl»

Periodicamente dovrebbe essere avviata una procedura per l'eliminazione degli articoli troppo vecchi. Questo viene attuato attraverso il programma '**expire**', che di solito viene avviato tramite uno script. Il file di configurazione di '**expire**' è '`/etc/news/expire.ctl`'.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all'interno della pagina di manuale *expire.ctl(5)*. Le righe vuote, quelle bianche e quelle che iniziano con il simbolo '**#**' vengono ignorate. Le direttive sono composte da record di due tipi, secondo le sintassi seguenti:

`/remember/:` ***n_giorni***

modello `:M|U|A:` ***n_giorni_min*** `:` ***n_giorni_predefinito*** `:` ***n_giorni_max***

La prima delle due sintassi si riferisce al tempo in giorni durante il quale si deve conservare memoria delle stringhe di identificazione degli articoli che sono passati per il sito; in pratica si tratta del contenuto del campo '**Message-ID:**' di ogni articolo. Questa indicazione è molto importante e la durata in questione non può essere troppo breve se non si vuole rischiare di ricevere nuovamente un articolo che in precedenza è già stato visto.

La seconda forma si riferisce ai record successivi. Con questi è possibile distinguere i tempi di scadenza degli articoli in base al gruppo a cui sono stati destinati ed eventualmente anche al fatto che questi siano moderati o meno. Per questo, nel secondo elemento si indica una lettera e precisamente: '**M**' identifica i gruppi moderati, '**U**' quelli non moderati e '**A**' tutti i gruppi senza distinguere su questo particolare.

Gli ultimi tre elementi delimitano la durata minima e massima di validità degli articoli; in particolare il valore intermedio è quello predefinito nel caso in cui l'articolo non disponga di questa informazione.

Si osservi l'esempio seguente:

```
/remember/:21
*:A:7:10:14
*:M:14:17:21
test*:A:1:1:1
```


Bisogna considerare che i gruppi rientrano sotto il controllo dell'ultimo record che coincide. In questo caso: le stringhe di identificazione degli articoli vengono conservate per 21 giorni; tutti i gruppi vengono conservati per un minimo di sette giorni, fino a un massimo di 14 (con un valore predefinito di 10); però i gruppi moderati sono conservati più a lungo (da 14 a 21 giorni); ma i gruppi che iniziano per '**test**' sono conservati solo un giorno. Sarebbe stato molto diverso se l'ordine fosse il seguente:

```
/remember/:21
test*:A:1:1:1
*:M:14:17:21
*:A:7:10:14
```

In questo caso, tutti i gruppi verrebbero conservati per un minimo di sette giorni, fino a un massimo di 14, compresi quelli che iniziano per '**test**'.

File «/etc/news/nnrp.access»

Il demone '**innd**' si avvale a sua volta di '**nnrpd**' per le connessioni con i programmi clienti (attraverso il protocollo NNTP) che si limitano a consultare gli articoli e a spedirne di nuovi. Il file di configurazione di '**nnrpd**' è '`/etc/news/nnrp.access`' con il quale si regolano questi accessi.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all'interno della pagina di manuale *nnrp.access(5)*. Le righe vuote, quelle bianche e quelle che iniziano con il simbolo '**#**' vengono ignorate. Le direttive sono composte da record secondo la sintassi seguente:

modello_nodi `:` ***permessi*** `:` [***utente***] `:` [***parola_d'ordine***] `:` ***modello_gruppi***

Il primo elemento permette di rappresentare un gruppo di nodi che possono accedere attraverso i caratteri jolly di INN. Il secondo serve a indicare i permessi di accesso, che sono costituiti dalla possibilità di leggere gli articoli (in questo caso si usa la lettera '**R**', *read*) e dalla possibilità di spedire degli articoli (lo si rappresenta con la lettera '**P**', *post*). Il terzo e il quarto elemento, se utilizzati, permettono di indicare un nominativo-utente e una parola d'ordine in chiaro. Il quinto elemento permette di individuare i gruppi a cui ci si riferisce, attraverso l'uso dei soliti caratteri jolly.

Come al solito, viene preso in considerazione l'ultimo record corrispondente all'accesso che viene tentato, per cui conviene mettere prima i record generici e alla fine quelli più dettagliati. In generale, bisogna evitare di concedere l'accesso a tutti, tanto che il file di configurazione predefinito viene fornito come si vede nell'esempio seguente:

```
# Default to no access
*:: -no- : -no- :!*
# Allow access from localhost
localhost:RP:::*
```

In pratica, si vieta espressamente l'accesso indiscriminato attraverso il record

```
*:: -no- : -no- :!*
```

dove quel '**-no-**' '**-no-**' è solo un modo appariscente per far capire che si tratta di una politica assolutamente sconsigliabile, per cui si concede l'accesso (sia per la lettura che per la spedizione di articoli) solo al nodo locale.

```
localhost:RP:::*
```

In sostituzione di questo record predefinito si potrebbe concedere l'accesso a tutta la propria rete locale, in un modo simile a quello seguente:

```
*.brot.dg:RP:::*
```

L'esempio seguente mostra in particolare un record con cui si concede l'accesso a qualunque nodo per la lettura dei gruppi '**comp.os.linux.***'.

```
*:R:::comp.os.linux.*
```

L'accesso può essere limitato in base all'indicazione di un nominativo-utente e di una parola d'ordine, come nell'esempio seguente:

```
*.brot.dg:RP:::*
*:RP:ignoto:segreto:*
```

In questo caso, l'idea è quella di permettere l'accesso indiscriminato dai nodi appartenenti al dominio *brot.dg* e di concederlo anche all'esterno, a patto che si fornisca il nominativo '**ignoto**' e la parola d'ordine '**segreto**'.

Evidentemente, se il file '`/etc/news/nnrp.access`' contiene l'indicazione di accessi controllati da una parola d'ordine, è necessario che non sia concessa la lettura di questo file agli utenti comuni.

File «/var/lib/news/active»

Inizialmente, sono disponibili alcuni gruppi amministrativi ('**control**', '**junk**', '**to**') e uno di prova, '**test**'. Per fare qualche prova, ciò è più che sufficiente. Volendo aggiungere qualche gruppo si potrebbe modificare il file '`/var/lib/news/active`', anche se per questo sarebbe meglio utilizzare il programma '**ctlinnd**' che viene descritto in seguito. È opportuno comunque conoscere il contenuto di questo file, che può contenere solo righe composte da quattro elementi secondo la sintassi seguente:

nome_del_gruppo n_iniziale n_finale opzione

La cosa migliore per cominciare è dare un'occhiata alla situazione iniziale.

```
control 0000000000 0000000001 y
junk 0000000000 0000000001 y
test 0000000000 0000000001 y
to 0000000000 0000000001 y
```

Il primo elemento rappresenta il nome del gruppo, il secondo rappresenta il numero attuale degli articoli presenti e il terzo indica il numero successivo. Per esempio, se si leggesse

```
test 0000000010 0000000011 y
```

significherebbe che nella directory '`/var/spool/news/test/`' c'è, o c'è stato, il file '`10`', mentre il prossimo articolo in questo gruppo verrebbe inserito nel file '`11`'.

L'ultimo elemento serve a stabilire il funzionamento del gruppo. La lettera '**y**' rappresenta un gruppo per il quale sono ammesse le spedizioni di articoli da parte dei clienti; in pratica rappresenta la situazione più comune. Per conoscere le altre opzioni disponibili e il loro significato si può consultare la pagina di manuale *active(5)*, comunque vengono riepilogate nella tabella u55.26.

Tabella u55.26. Elenco delle opzioni riferite ai gruppi all'interno del file '`active`'.

Opzione	Descrizione
`y`	È ammessa la lettura e la spedizione di articoli.
`n`	È ammessa solo la lettura degli articoli.
`x`	Il gruppo è disabilitato localmente.
`j`	Il gruppo non viene gestito.
`m`	Il gruppo è moderato e le spedizioni devono essere approvate.
`=`***gruppo***	Gli articoli vengono spostati nel gruppo indicato.

Come già accennato, inizialmente è meglio modificare questo file solo attraverso il programma '**ctlinnd**'.

File «/var/lib/news/history»

L'archivio storico degli articoli che sono stati visti viene conservato nel file '`/var/lib/news/history`' e in altri file con la stessa radice e con un'estensione particolare ('`history.*`'). Generalmente, questo deve essere creato la prima volta che si installa INN. Si procede semplicemente nel modo seguente:


```
# su news [Invio]
```

Si ottengono i privilegi dell'utente '**news**', dal momento che devono essere creati file che appartengono a questo nome.

```
news$ touch /var/lib/news/history [Invio]
```

Viene creato il file '`/var/lib/news/history`' vuoto.

```
news$ /usr/lib/news/bin/makehistory -ro [Invio]
```

Crea gli altri file abbinati per la gestione dell'archivio storico e l'operazione è conclusa.

Demoni e altri programmi per l'uso minimo di INN

Dopo aver definito una configurazione minima, anche senza aver aggiunto alcun gruppo a quelli predefiniti, si può fare qualche esperimento con l'uso di un cliente come Netscape o qualcosa di simile. Prima però occorre avviare il servizio NNTP.

Avvio e conclusione del servizio NNTP

Il servizio NNTP è gestito principalmente dal demone '**innd**' che, per quanto riguarda gli accessi da parte di clienti per la lettura e la spedizione di articoli, si avvale a sua volta di '**nnrpd**'. In pratica, a seconda della situazione, può capitare di vedere funzionare solo '**innd**', oppure anche una o più copie di '**nnrpd**' come sottoprocessi controllati sempre da '**innd**'. All'inizio del capitolo si è accennato al fatto che normalmente '**innd**' viene avviato attraverso uno script che potrebbe chiamarsi '**rc.news**' e si trova probabilmente nella directory '`/etc/rc.d/`'. È già stato spiegato anche che conviene dargli un'occhiata ed eventualmente può essere il caso di modificarlo. Oltre a '**innd**', questo script dovrebbe avviare '**innwatch**' per controllare che il sistema di news non superi lo spazio disponibile nel file system. In pratica, una volta avviato il servizio, si potrebbero osservare questi processi:

```
init-+-…
     |-…
     |-innd---2*[nnrpd]
     |-…
     |-rc.news---innwatch---sleep
     |-…
     `-…
```

Per avviare il servizio NNTP attraverso lo script '**rc.news**' occorre accedere con i privilegi dell'utente '**news**'.

```
news$ /etc/rc.d/rc.news [Invio]
```

Per disattivare il servizio, si uilizza un programma apposito per inviare un comando adatto a '**innd**': si tratta di '**ctlinnd**'. Nell'esempio mostrato sotto, prima viene inviato il comando '**throttle**' per bloccare il servizio, quindi viene inviato il comando '**shutdown**' per fare in modo che '**innd**' concluda del tutto il suo lavoro.

```
news$ /usr/lib/news/bin/ctlinnd throttle 'blocco del servizio' [Invio]

news$ /usr/lib/news/bin/ctlinnd shutdown 'chiusura del servizio' [Invio]
```

Dal momento che lo script '**rc.news**' aveva avviato anche '**innwatch**', occorre preoccuparsi di eliminare anche questo processo, per esempio nel modo seguente:

```
news$ killall innwatch [Invio]
```

Per semplificare tutto questo, la propria distribuzione GNU dovrebbe avere organizzato uno script aggiuntivo da collocarsi all'interno di '`/etc/rc.d/init.d/`' o in una posizione simile, allo scopo di poter avviare e concludere il servizio in modo più semplice:

```
/etc/rc.d/init.d/innd start|stop
```

Le prime volte è probabile che il servizio non si avvii, a causa di errori di configurazione. Evidentemente è necessario osservare i file delle registrazioni per vedere se appare la segnalazione della ragione per cui '**innd**' non parte. Spesso si tratta di file mancanti o di

errori nei permessi dei file che non consentono l'accesso all'utente di sistema '**news**'.

Utilizzo di «ctlinnd»

Il programma '**ctlinnd**' è uno dei pochi che potrebbe risultare accessibile nell'ambito dei percorsi normali di ricerca degli eseguibili. In pratica, potrebbe esserci un collegamento simbolico nella directory '`/usr/sbin/`' che permette di avviarlo senza dover indicare il percorso ('`/usr/lib/news/bin/`').

`ctlinnd` [*opzioni*] *comando* [*argomenti_del_comando*]

'**ctlinnd**' serve solo a inviare un comando a '**innd**', il quale risponde e l'esito determina il modo in cui '**ctlinnd**' termina. Generalmente si ottiene un '**Ok**' se tutto va bene, salvo alcuni comandi per i quali non viene generata alcuna risposta. I tipi di comando che possono essere usati sono molti e qui ne vengono descritti solo alcuni. Per conoscere l'uso dettagliato di '**ctlinnd**' conviene consultare la pagina di manuale *ctlinnd(8)*.

Comando	Descrizione
`pause` *motivazione*	Il comando '**pause**' serve a impedire le nuove connessioni, pur mantenendo quelle esistenti. Subito dopo viene chiuso l'archivio storico. L'argomento di questo comando è una stringa che serve a spiegare la ragione, in modo che possa essere annotata nel registro del sistema.
`throttle` *motivazione*	Il comando '**throttle**' serve a chiudere le connessioni esistenti e a rifiutarne delle nuove. Subito dopo viene chiuso l'archivio storico. L'argomento di questo comando è una stringa che serve a spiegare la ragione, in modo che possa essere annotata nel registro del sistema.
`go` [*motivazione*]	Questo comando viene usato dopo aver utilizzato '**pause**' o '**throttle**' per riaprire l'archivio storico e consentire nuovamente le connessioni. La stringa di motivazione dovrebbe coincidere con quella utilizzata per interrompere il servizio. Il comando '**go**' può essere usato per ripristinare il servizio dopo altri tipi di comandi, come descritto all'interno di *ctlinnd(8)*.
`shutdown` *motivazione*	Il comando '**shutdown**' serve a chiudere il servizio NNTP. Di solito è preferibile utilizzarlo dopo un comando '**throttle**'. L'argomento di questo comando è una stringa che serve a spiegare la ragione, in modo che possa essere annotata nel registro di sistema.
`newgroup` *nome_gruppo* [*opzione* [*creatore*]]	Il comando '**newgroup**' permette di creare un nuovo gruppo localmente. L'opzione si riferisce a ciò che può essere messo nel quarto elemento dei record del file '`/var/lib/news/active`'; se non viene specificato, si tratta della lettera '**y**' che abilita l'uso normale. L'ultimo argomento è il nome del creatore del gruppo. La creazione del gruppo aggiorna il file '`/var/lib/news/active`' e genera le directory necessarie a partire da '`/var/spool/news/`'.
`rmgroup` *nome_gruppo*	Questo comando elimina un gruppo attraverso la modifica del file '`/var/lib/news/active`'. La directory del gruppo eliminato non viene toccata e si lascia fare alla procedura di eliminazione degli articoli scaduti.

Segue la descrizione di alcuni esempi.

- `news$` **`/usr/lib/news/bin/ctlinnd throttle 'blocco del servizio'`** [*Invio*]

  Blocca il servizio NNTP senza chiudere il funzionamento di '**innd**'.

- `news$` **`/usr/lib/news/bin/ctlinnd shutdown 'chiusura del servizio'`** [*Invio*]

  Blocca il servizio NNTP e termina il funzionamento di '**innd**'.

- `news$` **`/usr/lib/news/bin/ctlinnd newgroup prova.discussioni.varie`** [*Invio*]

  Crea il gruppo '**prova.discussioni.varie**' e gli attribuisce l'opzione '**y**' in modo predefinito.

- `news$` **`/usr/lib/news/bin/ctlinnd rmgroup prova.discussioni.varie`** [*Invio*]

  Elimina il gruppo '**prova.discussioni.varie**' senza eliminare materialmente gli articoli ancora esistenti.

Operazioni di routine

L'eliminazione degli articoli troppo vecchi, secondo quanto configurato con il file '`/etc/news/expire.ctl`', viene fatta dal programma '**expire**', che però viene avviato solitamente tramite lo script '**news.daily**'. In pratica, attraverso il sistema Cron viene avviato giornalmente un comando come quello seguente:

```
su - news -c "/usr/lib/news/bin/news.daily"
```

Eventualmente, '**news.daily**' viene avviato con qualche opzione, come nel caso seguente:

```
su - news -c "/usr/lib/news/bin/news.daily delayrm"
```

Lo script '**news.daily**' serve anche per sistemare i file delle registrazioni (che dovrebbero trovarsi nella directory '`/var/log/news/`'), provvedendo alla loro rotazione, oltre che per avvisare l'amministratore del servizio, cioè l'utente '**news**', per mezzo della posta elettronica. '**news.daily**' accetta delle opzioni nella riga di comando, composte da delle parole chiave:

`news.daily` [*opzione*]…

Gli argomenti possibili sono molti e qui vengono descritte solo alcune delle opzioni. Eventualmente si può consultare la pagina di manuale *news.daily(8)*.

Opzione	Descrizione
`delayrm`	Ritarda la cancellazione degli articoli, accumulandoli in un file temporaneo.
`nostat`	Generalmente '**news.daily**' genera una serie di informazioni dettagliate sullo stato del sistema delle news. Con questa opzione si evita tale elaborazione.
`notdaily`	Se si vuole avviare '**news.daily**' al di fuori della sua cadenza giornaliera normale, conviene farlo con questa opzione, in modo che le operazioni di rotazione e archiviazione dei file delle registrazioni non siano svolte (assieme ad altre operazioni simili legate alla temporizzazione normale del suo utilizzo).
`noexpire`	Generalmente '**news.daily**' utilizza il programma '**expire**' per eliminare gli articoli troppo vecchi. Con questa opzione gli articoli non vengono rimossi.
`norotate`	In condizioni normali '**news.daily**' archivia ed esegue la rotazione dei file delle registrazioni. Con questa opzione non tocca i file delle registrazioni.

Feed in ingresso utilizzando il protocollo NNTP

Il feed degli articoli può avvenire in diversi modi, sia dal punto di vista del protocollo utilizzato, sia per il modo in cui viene temporizzato. In generale, attraverso Internet (o le intranet) si usa prevalentemente il protocollo NNTP. INN controlla il feed in ingresso attraver-

so il file ‘/etc/news/incoming.conf’, oppure, se si tratta di una versione più vecchia, ‘/etc/news/hosts.nntp’.

File «/etc/news/hosts.nntp»

Il file di configurazione ‘/etc/news/hosts.nntp’ riguarda le versioni di INN più vecchie. Serve a definire quali siano i nodi remoti che possono diffondere gli articoli verso il sistema di news locale.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all’interno della pagina di manuale *hosts.nntp(5)*. La presenza di questa pagina di manuale lascia intendere che sia ancora necessario l’utilizzo di questo file. Le righe vuote, quelle bianche e quelle che iniziano con il simbolo ‘#’ vengono ignorate. Le direttive sono composte da record secondo la sintassi seguente:

nodo : [[*parola_d’ordine*] : [*elenco_modelli_di_gruppi*]]

Considerato che si tratta di un file superato, non vale la pena di descriverne i dettagli. Basti sapere che per consentire la connessione è sufficiente indicare il nome del nodo seguito da due punti.

```
weizen.mehl.dg:
```

L’esempio mostra il caso in cui ci si attenda di avere il feed esclusivamente dal nodo *weizen.mehl.dg*. Eventualmente, ammesso che possa servire a qualcosa, si può aggiungere anche il nome del nodo locale:

```
localhost:
dinkel.brot.dg:
weizen.mehl.dg:
```

File «/etc/news/incoming.conf»

Il file di configurazione ‘/etc/news/incoming.conf’ riguarda le versioni di INN più recenti. Serve a definire i nodi remoti che possono diffondere gli articoli verso il sistema di news locale, oltre che stabilire il numero massimo di connessioni che possono instaurarsi simultaneamente.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all’interno della pagina di manuale *incoming.conf(5)*; la presenza di questa pagina di manuale fa intendere che sia necessario l’utilizzo del file di configurazione relativo. Le righe vuote, quelle bianche e quelle che iniziano con il simbolo ‘#’ vengono ignorate. Le direttive possono essere di vario tipo, ma soprattutto possono essere suddivise in sezioni ‘**peer**’ e ‘**group**’. Piuttosto di analizzare in dettaglio la sintassi di questo file, viene mostrato un esempio che dovrebbe essere sufficiente per iniziare.

```
# Definisce in modo globale il numero massimo di connessioni: 10
max-connections: 10

# Definisce l’accesso da parte dell’elaboratore weizen.mehl.dg
peer weizen {
    hostname:   weizen.mehl.dg
}
```

Nell’esempio appena mostrato sono state definire solo due cose: il numero massimo di connessioni in generale, fissando il valore a 10, e il fatto che *weizen.mehl.dg* può inviarci il suo feed di articoli.

Feed continuo in uscita utilizzando il protocollo NNTP

Il feed in uscita rappresenta il flusso di articoli che viene diffuso presso i nodi corrispondenti. Questo può avvenire fondamentalmente in modo continuo, attraverso ‘**innfeed**’, o in modo differito a cadenza regolare, attraverso ‘**nntpsend**’. ‘**innfeed**’ viene avviato normalmente da ‘**innd**’ in base alla configurazione del file ‘/etc/newsfeeds’.

Nelle prossime sezioni viene descritto cosa fare per utilizzare ‘**innfeed**’ nelle connessioni continue, ovvero di tipo a flusso (*stream*).

File «/etc/news/innfeed.conf»

Dovendo utilizzare ‘**innfeed**’ per la diffusione degli articoli, è necessario predisporre il file ‘/etc/news/innfeed.conf’. Questo dovrebbe essere già stato predisposto abbastanza bene da chi ha preparato il pacchetto INN da installare.

La sintassi e le direttive che possono essere utilizzate in questo file sono descritte all’interno della pagina di manuale *innfeed.conf(5)*; come al solito, le righe vuote, quelle bianche e quelle che iniziano con il simbolo ‘#’, vengono ignorate.

Se tutto va bene, si dovrebbe porre attenzione solo alla dichiarazione dei nodi a cui inviare gli articoli per la loro diffusione. Per esempio, la direttiva

```
peer schwarz {
    hostname:   schwarz.mehl.dg
}
```

identifica il nodo *schwarz.mehl.dg* e gli attribuisce il nomignolo ‘**schwarz**’ che serve poi per definire la duplicazione degli articoli per questo scopo nel file ‘/etc/news/newsfeeds’.

Il tipo di connessione che si intende mostrare qui è di tipo a flusso continuo (*stream*), di conseguenza, prima della dichiarazione dei nodi dovrebbe apparire la direttiva seguente:

```
streaming:      true
```

Eventualmente si può essere sicuri ripetendola nella dichiarazione del nodo:

```
peer schwarz {
    hostname:   schwarz.mehl.dg
    streaming:  true
}
```

File «/etc/news/newsfeeds»

Come già accennato, per fare in modo che ‘**innfeed**’ venga avviato da ‘**innd**’ nel modo corretto, occorre predisporre opportunamente il file ‘/etc/news/newsfeeds’. In precedenza è stato mostrato solo come attivare la diffusione locale degli articoli, per mezzo della voce standard ‘**ME**’; adesso occorre indicare che è necessario diffondere gli articoli attraverso ‘**innfeed**’:

```
# Innfeed funnel master; individual peers feed into the funnel.
# Note that innfeed with "-y" and no peer in innfeed.conf
# would cause a problem that innfeed drops the first article.
innfeed!:\
        !*\
        :Tc,Wnm*:/usr/lib/news/bin/startinnfeed
```

Di solito, la direttiva che si vede nell’esempio è già contenuta nel file standard che viene installato con INN; eventualmente si tratta solo di togliere i commenti che ne impediscono l’attivazione.

Senza entrare troppo nel dettaglio (che comunque può essere approfondito con la lettura di *newsfeeds(5)*), si può affermare che viene creato un feed attraverso un condotto. Questo, come si legge dal commento originale, viene definito «imbuto» (*funnel*). Osservando bene, si vede che nel secondo elemento è indicato il modello ‘**!***’, cosa che impedisce la corrispondenza con qualunque articolo; infatti occorre indicare espressamente quali nodi alimentare in questo modo attraverso altre direttive successive.

```
# A real-time feed through innfeed.
schwarz\
        :!junk,!control,!test,!local*\
        :Tm:innfeed!
```

Come si vede dall’esempio, viene creato un feed verso il nodo indicato nel file ‘/etc/news/innfeed.conf’ con il nomignolo di ‘**schwarz**’, per tutti i gruppi che non siano ‘**junk**’, ‘**control**’, ‘**test**’ e nemmeno che inizino per ‘**local**’. Questo flusso viene incanalato verso ‘**innfeed**’ attraverso la direttiva denominata ‘**innfeed!**’ (quella di prima).

Evidentemente, dovendo fare il feed nello stesso modo verso altri nodi, basterebbe aggiungere altre direttive di questo tipo che si rivolgono sempre alla voce ‘**innfeed!**’.

Per riepilogare un po’, viene mostrato un esempio complessivo che comprende anche una dichiarazione ipotetica della diffusione locale.

```
ME:*,!junk,!control*,!local*::

innfeed!:!*:Tc,Wnm*:/usr/lib/news/bin/startinnfeed

schwarz:!junk,!control,!test,!local*\
        :Tm:innfeed!
```

Feed periodico in uscita utilizzando il protocollo NNTP

Per fare il feed periodico in uscita attraverso il protocollo NNTP, si utilizza il programma '**innxmit**', di solito attraverso lo script '**nntpsend**'. Per ottenere tale risultato è opportuno predisporre il file '`/etc/news/nntpsend.ctl`' con l'elenco dei nodi che si vogliono servire in questo modo, quindi è necessario predisporre nel file '`/etc/news/newsfeeds`' la dichiarazione di questa forma di feed per ognuno di questi nodi.

File «/etc/news/nntpsend.ctl»

Il file '`/etc/news/nntpsend.ctl`' contiene la configurazione per lo script '**nntpsend**' e serve a elencare i nodi ai quali si vuole inviare il feed a intervalli regolari, attraverso il programma '**innxmit**'.

Le direttive di questo file sono dei record e la sintassi relativa può essere approfondita leggendo la pagina di manuale *nntpsend.ctl(5)*.

```
##  Control file for nntpsend.
## Format:
##      site:fqdn:max_size:[<args...>]
##      <site>          The name used in the newsfeeds file for this site;
##                      this determines the name of the batchfile, etc.
##      <fqdn>          The fully-qualified domain name of the site,
##                      passed as the parameter to innxmit.
##      <size>          Size to truncate batchfile if it gets too big;
##                      see shrinkfile(1).
##      <args>          Other args to pass to innxmit
##  Everything after the pound sign is ignored.

heiden:heiden.mehl.dg::
```

L'esempio, che riporta anche i commenti originali del file, mostra un record con il quale si vuole definire il feed verso il nodo *heiden.mehl.dg*, identificato ai fini della configurazione con il nomignolo '**heiden**'.

File «/etc/news/newsfeeds»

È necessario intervenire anche nel file '`/etc/news/newsfeeds`' per convogliare una copia degli articoli verso ogni nodo per il quale si utilizza questa forma differita di diffusione. L'esempio seguente dichiara il feed verso un file che poi viene letto da '**innxmit**' per l'invio verso il nodo *heiden.mehl.dg*, identificato nel file di configurazione '`/etc/news/nntpsend.ctl`' con il nomignolo '**heiden**'.

```
# Feed all local non-internal postings to nearnet; sent off-line via
# nntpsend or send-nntp.
amico-mio\
        :!junk,!control,!test,!local*\
        :Tf,Wnm:heiden
```

Si osservi che nel primo elemento del record è stato usato un nome di fantasia per identificare la voce, mentre l'ultimo fa riferimento al nomignolo fissato nel file '`/etc/news/nntpsend.ctl`'.

Per essere precisi, in questo caso viene creato un file nella directory '`/var/spool/news/out.going/`' con i riferimenti agli articoli da usare per il feed, che poi viene letto da '**nntpsend**' quando è il momento di fare il trasferimento.

Utilizzo di «nntpsend»

Lo script '**nntpsend**' è il mezzo più comodo per comandare il programma '**innxmit**' allo scopo di fare il feed per mezzo del protocollo NNTP. Se viene utilizzato senza argomenti, '**nntpsend**' legge il file di configurazione '`/etc/news/nntpsend.ctl`' e diffonde gli articoli verso i nodi che vi trova elencati, in base al contenuto dei file relativi accumulati in precedenza in base alla configurazione di '`/etc/news/newsfeeds`'.

Questo, come tutto ciò che riguarda INN, deve essere avviato con i privilegi dell'utente '**news**':


```
news$ /usr/lib/news/bin/nntpsend [ Invio ]
```

Se si utilizza questa forma di diffusione degli articoli, conviene predisporre il sistema Cron al riguardo, eventualmente attraverso uno script simile a quello seguente:

```
#!/bin/sh
su - news -c /usr/lib/news/bin/nntpsend
```

Ritrasmissione di articoli attraverso la posta elettronica

Una forma alternativa di feed è la trasmissione di una copia degli articoli verso un indirizzo di posta elettronica. Si ottiene questo semplicemente inserendo le direttive necessarie nel file '`/etc/news/newsfeeds`'. Per la precisione, si deve definire un imbuto, per esempio la direttiva seguente:

```
# Imbuto per l'invio attraverso la posta elettronica.
mailer!:!*:Tp,W*:/bin/mail -s "Articoli da Usenet" *
```

Come si vede, viene utilizzato '`/bin/mail`' a cui viene aggiunta l'indicazione dell'oggetto («Articoli di Usenet») e l'indirizzo è rappresentato dall'asterisco finale. Per ottenere effettivamente l'invio dei messaggi occorre indicare altre direttive, una per ogni indirizzo, che utilizzano l'imbuto appena creato.

```
# Spedisce i gruppi comp.os.linux e alt.comp.os.linux a
# tizio@dinkel.brot.dg
tizio@dinkel.brot.dg:!*,comp.os.linux,alt.comp.os.linux:Tm:mailer!

# Spedisce il gruppo it.cultura.linguistica.italiano a
# caio@dinkel.brot.dg
caio@dinkel.brot.dg:!*,it.cultura.linguistica.italiano:Tm:mailer!
```

Si osservi in particolare che nel secondo elemento di questi record viene indicato inizialmente di escludere tutti i gruppi, con il modello '**!***', quindi di includere ciò che si desidera. Se non si facesse così, si otterrebbe l'invio degli articoli di tutti i gruppi.

Prelievo di articoli utilizzando il protocollo NNTP

Il prelievo di articoli non dovrebbe essere una tecnica usuale per ottenere il feed da un sito remoto, però potrebbe essere utile quando l'accesso a Internet è fatto attraverso una linea commutata: nel momento in cui si apre questa linea, oltre che inviare gli articoli prodotti nella rete locale, si vogliono ricevere quelli nuovi provenienti dall'esterno. Questo prelievo si può ottenere attraverso il programma '**nntpget**'.

Utilizzo di «nntpget»

Il programma '**nntpget**' non dispone di un file di configurazione ed è fatto per essere gestito comodamente attraverso degli script esterni, che però probabilmente sono mancanti. Come si vede dalla sintassi, a parte le opzioni che in pratica sono necessarie, è indispensabile indicare il nodo dal quale prelevare gli articoli aggiornati.

```
nntpget [opzioni] nodo
```

Il programma '**nntpget**' va visto probabilmente solo come compendio al sistema locale di gestione delle news; in tal senso è praticamente necessario che sia in funzione il demone '**innd**', in modo che '**nntpget**' possa sapere quali articoli caricare e quali no. Si osservi l'esempio seguente:

```
news$ /usr/lib/news/bin/nntpget -o -v -t '990324 000000' ↩
↪roggen.brot.dg [ Invio ]
```

L'opzione '**-o**' richiede espressamente la comunicazione con il demone '**innd**' per conoscere quali articoli vale la pena di caricare dal sito remoto; l'opzione '**-v**' fa in modo di avere qualche informazione in più; l'opzione '**-t '990324 000000'**' fa in modo che vengano cercati solo gli articoli più recenti rispetto all'ora zero del 24/03/1999; l'ultimo argomento indica di contattare il nodo *roggen.brot.dg*.

In questa situazione, l'indicazione di una data di riferimento attraverso l'opzione '**-t**' è obbligatoria e il formato è stabilito dal servente:

AAMMGG HHMMSS

In pratica: anno, mese, giorno, spazio, ore, minuti, secondi.

Consultando la pagina di manuale di '**nntpget**' si può leggere in che modo sostituire l'opzione '**-t**' con '**-f**', allo scopo di usare un file al posto della data, sfruttando la sua data di modifica come riferimento per il prelievo degli articoli.

Utilizzando '**nntpget**' in questo modo, è necessario che il servente che viene contattato consenta l'uso del comando '**NEWNEWS**', che forse deve essere abilitato espressamente. Con le versioni recenti di INN occorre la direttiva '**allownewnews true**' nel file '`/etc/news/inn.conf`'.

Replicazione dei gruppi di un altro sito

Fino a questo punto si è visto che per creare un gruppo si può utilizzare il comando '**ctlinnd newgroup** ***nome***'. In alternativa si può intervenire direttamente nel file '`/var/lib/news/active`', ma poi c'è il problema di creare fisicamente le directory che devono ospitare gli articoli. Per preparare rapidamente un sito Usenet, può essere conveniente il prelievo di una copia di questo file da uno dei siti corrispondenti attraverso il programma '**actsync**'.

'**actsync**' viene configurato attraverso il file '`/etc/news/actsync.cfg`' e si avvale generalmente di '`/etc/news/actsync.ign`' per stabilire quali sono i gruppi da ignorare e quali da tenere. Per conoscere i dettagli sul funzionamento di '**actsync**' e sul modo di configurarlo attraverso i file citati, occorre leggere la pagina di manuale *actsync(8)*. A titolo informativo sulle possibilità di '**actsync**' vengono mostrati un paio di esempi.

```
news$ /usr/lib/news/bin/actsync -o a news.brot.dg [Invio]
```

Il comando mostrato sopra, permette di accedere al servizio NNTP di *news.brot.dg*, emettendo attraverso lo standard output un risultato simile a quello seguente, che in pratica riproduce un file '`active`', ottenuto togliendo i gruppi da escludere in base alla configurazione di '**actsync**'.

```
comp.os.linux 0000000002 0000000123 y
alt.comp.os.linux 0000000145 0000000345 y
    …
```

Il comando seguente, invece di mostrare il contenuto del file '`active`', serve ad aggiornare i gruppi locali in base all'esito ottenuto. In pratica '**actsync**' si avvale di '**ctlinnd**' per questo.

```
news$ /usr/lib/news/bin/actsync -p 0 -o x -z 0 news.brot.dg
[Invio]
```

Riferimenti

- Olaf Kirch, *NAG, The Linux Network Administrators' Guide*
- Rich Salz, James Brister, *Installing InterNet News*
  '`/usr/share/doc/inn/Install.ms`'
- Ian Jackson, Miquel van Smoorenburg, *Configuring Debian GNU/Linux's INN package*
  '`/usr/share/doc/inn/…`'

Parte xx

Cfengine

Introduzione a Cfengine 331
- Primo approccio con la configurazione 331
- Sezioni e classi predefinite 332
- Classi più in dettaglio 334
- Variabili e stringhe 336
- Espressioni regolari 339

Cfengine: sezioni di uso comune 341
- Permessi e proprietà 341
- Sezione «control» 342
- Sezione «classes» o «groups» 343
- Sezione «copy» 343
- Sezione «directories» 345
- Sezione «disable» 345
- Sezione «files» 346
- Sezione «links» 346
- Sezione «processes» 348
- Sezione «shellcommands» 349
- Sezione «tidy» 349

Cfengine attraverso la rete 351
- Configurazione e avvio del demone 351
- Filosofia del sistema di distribuzione di Cfengine 353

Introduzione a Cfengine

Primo approccio con la configurazione 331
La variabile CFINPUTS 332
Simulazione 332
Sezioni e classi predefinite 332
Classi più in dettaglio 334
Variabili e stringhe 336
Elenchi 338
Espressioni regolari 339

Cfengine [1] è uno strano sistema di amministrazione di elaboratori Unix, la cui importanza si apprende solo con il tempo e con l'utilizzo. Il suo scopo è quello di facilitare l'amministrazione di tali sistemi operativi, soprattutto quando si dispone di un gruppo eterogeneo di questi su diversi elaboratori. Questi capitoli dedicati a Cfengine non pretendono di esaurire l'argomento, cercando piuttosto di semplificare il suo apprendimento che poi può essere approfondito leggendo la documentazione originale.

A prima vista, si può intendere Cfengine come l'interprete di un linguaggio molto evoluto. In questo capitolo si introduce l'uso specifico dell'eseguibile '**`cfengine`**', il cui scopo è interpretare un file di configurazione, ovvero il suo script, agendo di conseguenza.

Primo approccio con la configurazione

Per funzionare, l'eseguibile '**`cfengine`**' richiede la presenza di un file di configurazione, che eventualmente può essere trasformato in script, se ciò può essere conveniente. La comprensione, anche elementare, del modo in cui si configura questo programma, è la chiave per capire a cosa può servire in generale Cfengine.

Il file di configurazione di '**`cfengine`**' ha una struttura speciale, in cui però si possono inserire commenti, preceduti dal simbolo '`#`', e righe vuote o bianche. In particolare, a proposito dei commenti, se questi si collocano alla fine di una direttiva, devono essere staccati da questa con uno o più spazi orizzontali.

Le direttive del file di configurazione vanno inserite all'interno di sezioni ed eventualmente all'interno di classi. In altri termini, il file di configurazione si articola in sezioni, che possono contenere direttive o scomporsi in classi, che a loro volta contengono le direttive. Come si intende, la suddivisione in classi è facoltativa, ma si tratta comunque di una caratteristica fondamentale di Cfengine, in quanto consente di selezionare le direttive da prendere in considerazione in base all'appartenenza o meno dell'elaboratore alle classi stesse.

Dal momento che il problema non è semplice da esporre, conviene iniziare subito con un esempio che possa essere verificato senza troppi problemi anche da un utente comune:

```
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

Se questo file si chiama '`cfengine.conf`' e si trova nella directory corrente, qualunque essa sia, se non è stata impostata la variabile di ambiente '**`CFINPUTS`**', si può avviare l'interpretazione di tale file semplicemente avviando l'eseguibile '**`cfengine`**':

```
$ cfengine [ Invio ]
```

Quello che si ottiene è soltanto la creazione del collegamento simbolico '`/var/tmp/altra`' che punta in realtà alla directory '`/tmp/`'.

Se il file di configurazione fosse collocato altrove, eventualmente con un'altra denominazione, si potrebbe ottenere lo stesso risultato

con il comando seguente, dove il nome del file viene aggiunto nella riga di comando:

```
$ cfengine -f file_di_configurazione [Invio]
```

Infine, per realizzare uno script dalla configurazione, basta inserire all'inizio una riga simile a quella seguente (ammesso che l'eseguibile si trovi effettivamente in '`/usr/bin/`'):

```
#!/usr/bin/cfengine -f
```

In altri termini, lo script completo dell'esempio precedente sarebbe:

```
#!/usr/bin/cfengine -f
# Esempio di partenza

control:
    actionsequence = ( links )

links:
    /var/tmp/altra -> /tmp
```

La variabile CFINPUTS

La variabile di ambiente '**`CFINPUTS`**' serve per definire un percorso di ricerca per il file di configurazione. In generale, se si utilizza l'opzione '**`-f`**' specificando un percorso assoluto, a partire dalla radice (qualcosa che inizia con '`/`'), si tratta esattamente di quel file, altrimenti, se è disponibile la variabile '**`CFINPUTS`**', questa viene preposta al nome del file indicato. Per esempio, il comando

```
$ cfengine -f prova [Invio]
```

fa riferimento precisamente al file di configurazione '`$CFINPUTS/prova`', ovvero al file '`./prova`' se la variabile '**`CFINPUTS`**' non è disponibile.

Quando non si indica il file di configurazione, si fa implicitamente riferimento al nome '`cfengine.conf`'. In tal caso si tratta precisamente di '`$CFINPUTS/cfengine.conf`', ovvero del file '`./cfengine.conf`' in mancanza della variabile '**`CFINPUTS`**'.

Simulazione

Cfengine è un sistema molto potente, i cui script definiscono operazioni molto complesse con poche direttive. Di fronte a direttive distruttive occorre essere sicuri del risultato che si ottiene effettivamente. Per verificare cosa farebbe Cfengine con la configurazione stabilita, senza eseguire realmente la cosa, si può usare l'opzione '**`-n`**', abbinata a '**`-v`**': la prima simula l'esecuzione; la seconda mostra nel dettaglio cosa succede o cosa dovrebbe succedere.

Finché non si è sicuri del proprio script o della propria configurazione, occorre ricordare di fare tutte le prove utilizzando l'opzione '**`-n`**'.

Realizzando uno script con questo intento, basta modificare la prima riga nel modo seguente:

```
#!/usr/bin/cfengine -n -v -f
```

Sezioni e classi predefinite

Le direttive del file di configurazione vanno inserite all'interno di sezioni, che a loro volta possono suddividersi in classi. Le sezioni rappresentano dei tipi di azione e i loro nomi sono già stabiliti.

```
sezione_ovvero_tipo_di_azione :

    definizione_della_classe : :

        direttiva_o_azione
        ...

    ...
```

Negli esempi visti fino a questo punto, sono state mostrate le sezioni '**`control`**' e '**`links`**'. Nella sezione '**`control`**' è stata inserita la direttiva '**`actionsequence`**', che ha l'aspetto di un assegnamento a una variabile:

```
control:
    actionsequence = ( links )
```

Le direttive, ovvero le istruzioni che possono apparire all'interno di classi o di sezioni non suddivise in classi, possono occupare una o più righe, senza bisogno di simboli di continuazione e senza bisogno di simboli per la conclusione delle istruzioni stesse.

In questo caso particolare, si tratta di assegnare uno o più nomi, che rappresentano altrettante sezioni, alla sequenza di esecuzione. In pratica, la direttiva dell'esempio stabilisce che deve essere eseguita la sezione '**`links`**'. Se non venisse specificata in questo modo, la sezione '**`links`**' non verrebbe presa in considerazione. Pertanto, la configurazione seguente non produrrebbe alcunché:

```
# Non fa nulla

control:
    actionsequence = ( )

links:
    /var/tmp/altra -> /tmp
```

Il prossimo esempio dovrebbe chiarire definitivamente questo particolare. Si osservi il fatto che si vuole eseguire prima la sezione '**`tidy`**' e poi la sezione '**`links`**', anche se l'ordine in cui sono mostrate poi le sezioni è inverso.

```
control:
    actionsequence = ( tidy links )

links:
    /var/tmp/altra -> /tmp

tidy:
    /var/tmp pattern=* age=30 recurse=inf
```

In questo caso, la sezione '**`tidy`**' serve a programmare la cancellazione di file e directory. Per la precisione, la direttiva che si vede cancella tutti i file e le directory a partire da '`/var/tmp/`', purché la data di accesso sia trascorsa da almeno 30 giorni. Si osservi anche l'opzione '**`recurse=inf`**', che richiede una ricorsione infinita nelle sottodirectory. In condizioni normali, questa ricorsione non dovrebbe attraversare i collegamenti simbolici, mentre per ottenere tale comportamento occorrerebbe aggiungere l'opzione '**`-l`**'. Pertanto, anche se dovesse esistere già il collegamento simbolico '`/var/tmp/altra`', che punta a '`/tmp/`', questa directory non verrebbe scandita se non richiesto espressamente.

Le classi sono la caratteristica fondamentale di Cfengine, perché consentono di distinguere le direttive di una sezione in base a una sottoclassificazione che serve a selezionare un gruppo ristretto di elaboratori. In pratica, consente di indicare direttive differenti in base alla «classificazione» a cui appartengono gli elaboratori presi in considerazione. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )

links:
    linux_2.2.15::
        /var/tmp/altra -> /tmp
    linux_2.2.16::
        /var/tmp/altre -> /tmp
        /var/tmp/altri -> /tmp
```

Anche se poco significativo, l'esempio è abbastanza semplice e dovrebbe permettere di comprendere il senso della distinzione in classi. In questo caso, la sezione '**`links`**' si articola in due classi, denominate '**`linux_2.2.15`**' e '**`linux_2.2.16`**'. Se viene usato questo file in un elaboratore con un sistema GNU/Linux avente un kernel 2.2.15, si ottiene il collegamento simbolico '`/var/tmp/altra`', mentre con un kernel 2.2.16 si otterrebbero due collegamenti simbolici: '`/var/tmp/altre`' e '`/var/tmp/altri`'. Naturalmente, questa operazione può non avere molto significato in generale, ma l'esempio serve a mostrare la possibilità di indicare direttive diverse in base alla classe a cui appartiene l'elaboratore.

La classe serve principalmente a individuare il sistema operativo (nel caso di GNU/Linux si tratta del nome del kernel), in modo da cambiare azione in funzione delle consuetudini di ogni ambiente. In que-

sto caso, volendo selezionare un sistema GNU/Linux senza specificare la versione del kernel sarebbe stato sufficiente indicare la classe '**linux**'. Tuttavia, come si vede nell'esempio, esistono delle classi più dettagliate che permettono di raggiungere anche altre caratteristiche. Per conoscere quali sono le classi valide nell'elaboratore che si utilizza in un certo momento, basta il comando seguente:

```
$ cfengine -p -v [Invio]
```

A titolo di esempio, ecco cosa potrebbe comparire:

```
GNU Configuration Engine -
cfengine-1.5.3
Free Software Foundation 1995, 1996, 1997
Donated by Mark Burgess, Centre of Science and Technology
Faculty of Engineering, Oslo College, 0254 Oslo, Norway

-------------------------------------------------------------------------

Host name is: dinkel
Operating System Type is linux
Operating System Release is 2.2.15
Architecture = i586


Using internal soft-class linux for host dinkel

The time is now Tue Oct 24 16:11:18 2000


-------------------------------------------------------------------------

Additional hard class defined as: 32_bit
Additional hard class defined as: linux_2.2.15
Additional hard class defined as: linux_i586
Additional hard class defined as: linux_i586_2.2.15
Additional hard class defined as:
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000

GNU autoconf class from compile time: linux-gnu

  Careful with this - it might not be correct at run time if you have
  several OS versions with binary compatibility!

Address given by nameserver: 192.168.1.1
dinkel: No preconfiguration file
Accepted domain name: undefined.domain


Defined Classes = ( any debian linux dinkel undefined_domain Tuesday Hr16 Min11
 Min10_15 Day24 October Yr2000 32_bit linux_2_2_15 linux_i586 linux_i586_2_2_15
 linux_i586_2_2_15__1_Thu_Aug_31_15_55_32_CEST_2000 linux_gnu 192_168_1
 192_168_1_1 )

Negated Classes = ( )

Installable classes = ( )

Global expiry time for locks: 120 minutes

Global anti-spam elapse time: 0 minutes

Extensions which should not be directories = ( )
Suspicious filenames to be warned about = ( )
```

Le classi disponibili sono quindi quelle elencate nell'insieme '**Defined Classes**'. Si può osservare che è accessibile anche una classe con il nome della distribuzione GNU/Linux (in questo caso è Debian), oltre agli indirizzi IP abbinati all'interfaccia di rete.

Classi più in dettaglio

Le classi non sono necessariamente nomi singoli; possono essere delle espressioni composte da più nomi di classe, uniti tra loro attraverso operatori booleani opportuni. Prima di arrivare a descrivere questo, è bene riassumere le classi più comuni e vedere come si possono definire delle classi nuove. Una classe elementare può essere:

- la parola chiave '**any**', che rappresenta tutti gli elaboratori;
- il nome del sistema operativo o del kernel, assieme a una serie di varianti che includono altre caratteristiche dell'architettura del sistema;
- il nome finale dell'elaboratore (senza il dominio eventuale a cui appartiene);
- il nome che identifica una componente del tempo (giorno, ora, minuto, ecc.), come si vede nella tabella u56.10;
- il nome di un gruppo di classi definito per comodità dell'utilizzatore;

- il nome di una classe libera definito per comodità dell'utilizzatore.

Tabella u56.10. Elenco delle classi di Cfengine riferite al tempo.

Nome	Descrizione
Monday, Tuesday, Wednesday,...	Giorni della settimana.
Hr00, Hr01,... Hr23	Ore del giorno.
Min00, Min01,... Min59	Minuti di un'ora.
Min00_05, Min05_10,... Min55_00	Intervalli di cinque minuti.
Day1, Day2,... Day31	Giorni del mese.
January, February,... December	Mesi dell'anno.
Yr1999, Yr2000, Yr2001,...	Anni.

Si può definire un gruppo di classi attraverso la sezione '**classes**' o '**groups**', in cui le direttive servono per definire delle classi nuove raggruppando più classi preesistenti:

```
classes: | groups:
    gruppo_di_classi = ( classe_1 classe_2... )
    ...
```

Per esempio, la dichiarazione seguente serve a raggruppare in due classi nuove le ore del mattino e le ore della sera, supponendo che ciò possa avere un significato pratico di qualche tipo:

```
classes:
    OreDelMattino = ( Hr06 Hr07 Hr08 Hr09 )
    OreDellaSera  = ( Hr18 Hr19 Hr20 Hr21 )
```

Inoltre si possono definire delle classi in base al risultato soddisfacente di un programma o di uno script. In altri termini, se un programma restituisce *Vero*, questo fatto può essere preso in considerazione come motivo valido per generare una classe. L'esempio seguente crea la classe '**miashell**' se è presente il file '`/bin/bash`' oppure il file '`/bin/zsh`':

```
classes:
    miashell = ( "/bin/test -f /bin/bash" "/bin/zsh" )
```

Si possono dichiarare anche delle classi fittizie, il cui significato si può comprendere solo in un secondo momento. Queste classi fittizie si dichiarano nella sezione '**control**', con la direttiva '**addclasses**':

```
control:
    ...
    addclasses = ( classe_fittizia... )
    ...
```

L'esempio seguente crea due classi fittizie, denominate '**bianchi**' e '**rossi**':

```
    addclasses = ( bianchi rossi )
```

Avendo più chiaro in mente cosa possa essere una classe elementare, si può iniziare a descrivere la definizione di espressioni legate alle classi. Le espressioni in questione sono booleane, dal momento che le classi, di per sé, rappresentano degli insiemi di elaboratori. In questo senso, la logica booleana si intende correttamente come la logica degli insiemi. Gli operatori di queste espressioni sono elencati nella tabella u56.14.

Tabella u56.14. Operatori logici delle espressioni riferite alle classi di Cfengine.

Operatore	Descrizione
( )	Le parentesi tonde hanno la precedenza nella valutazione.
!	NOT, ovvero insieme complementare.
.	AND, ovvero intersezione.
|	OR, ovvero unione.

Operatore	Descrizione
| |	Modo alternativo di indicare OR.

Per esempio, per indicare una classe complessiva che rappresenta indifferentemente un elaboratore con sistema operativo GNU/Linux o GNU/Hurd, si può usare l'espressione '**linux|hurd**'. In pratica, si scrive così:

```
linux|hurd::
```

Per indicare una classe che rappresenti tutti gli elaboratori che non abbiano un sistema operativo GNU/Linux, si potrebbe usare l'espressione '**!linux**', ovvero:

```
!linux::
```

A questo punto diventa più facile comprendere il senso delle classi fittizie che si possono dichiarare con la direttiva '**addclasses**'. Si osservi l'esempio seguente:

```
control:
    actionsequence = ( links )
    addclasses = ( primo )

links:
    any.primo::
        /var/tmp/altra -> /tmp
```

L'espressione '**any.primo**' si avvera solo quando la classe elementare '**primo**' è stata dichiarata come nell'esempio; infatti, '**any**' è sempre vera. In questo modo, anche se l'esempio non richiederebbe tanta raffinatezza, basterebbe controllare la dichiarazione della direttiva '**addclasses**' per abilitare o meno la classe sottostante. In altri termini, è facile modificare un file di configurazione che richiama in più punti la classe fittizia '**primo**', modificando solo una riga di codice nella sezione '**control**'.

Il controllo sulla definizione di classi fittizie può avvenire anche al di fuori del file di configurazione attraverso le opzioni '**-D*classe_fittizia***' e '**-N*classe_fittizia***'. Nel primo caso, si ottiene la dichiarazione di una classe fittizia, mentre nel secondo si ottiene l'eliminazione di una classe già dichiarata nel file di configurazione. Per esempio, il comando seguente serve ad annullare l'effetto della dichiarazione della classe fittizia '**primo**', dell'esempio precedente.

```
$ cfengine -Nprimo -f prova.conf [Invio]
```

Variabili e stringhe

Cfengine gestisce le variabili di ambiente, oltre ad altre variabili, in modo simile a quanto fanno le shell. Queste variabili vengono espanse usando una delle due notazioni seguenti:

```
$(nome_variabile)
${nome_variabile}
```

Per la precisione, le variabili di Cfengine possono essere state ereditate dall'ambiente, possono essere state definite nella sezione '**control**', oppure possono essere variabili predefinite di Cfengine. L'esempio seguente mostra la dichiarazione della variabile '**percorso**' nella sezione '**control**':

```
control:
    actionsequence = ( tidy links )
    percorso = ( "/var/tmp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si intuisce che potrebbe essere più interessante dichiarare la variabile in questione all'interno di classi diverse, in modo da aggiornare automaticamente il percorso di conseguenza. L'esempio seguente mostra due classi inventate, '**bianco**' e '**nero**', che non esistono in realtà:


```
control:
    actionsequence = ( tidy links )
    bianco::
        percorso = ( "/var/tmp" )
    nero::
        percorso = ( "/temp" )

tidy:
    $(percorso) pattern=* age=30 recurse=inf

links:
    $(percorso)/altra -> /tmp
```

Si può osservare in particolare che la direttiva '**actionsequence**', non appartenendo ad alcuna classe, viene presa sempre in considerazione.

Le variabili predefinite di Cfengine sono tali perché sono gestite automaticamente e servono a rendere disponibili delle informazioni, oppure perché servono a definire delle informazioni specifiche. In altri termini, le prime vanno solo lette, mentre le altre vanno impostate opportunamente se richiesto. La tabella u56.20 mostra le variabili destinate alla sola lettura, mentre la tabella u56.21 mostra le variabili da impostare.

Tabella u56.20. Variabili interne di Cfengine, destinate alle sola lettura.

Variabile	Descrizione
`allclasses`	Elenca le classi attive.
`arch`	Architettura in modo dettagliato.
`binserver`	Servente NFS predefinito per dati binari.
`class`	Classe essenziale riferita al sistema operativo.
`date`	La data attuale.
`fqhost`	Il nome a dominio completo.
`ipaddress`	Un indirizzo IP significativo dell'elaboratore.
`year`	L'anno attuale.

Per quanto riguarda la variabile '**domain**', se questa non viene impostata espressamente, occorre considerare che potrebbe trattarsi del dominio che compone il nome dell'elaboratore, ovvero ciò che si legge e si imposta con il comando '**hostname**' dei sistemi Unix. In pratica, se il nome dell'elaboratore è stato impostato senza l'aggiunta del dominio di appartenenza, questa variabile restituisce probabilmente la stringa '**undefined.domain**'. Lo stesso discorso vale per la variabile '**fqhost**': se non si dispone del dominio finale nel nome restituito da '**hostname**', si ottiene una cosa simile a '***nome*.undefined.domain**'.

Tabella u56.21. Variabili interne di Cfengine, modificabili da parte dell'utilizzatore.

Variabile	Descrizione
`domain`	Il dominio, senza il nome iniziale dell'elaboratore.
`faculty` `site`	Nome utilizzabile per definire il luogo.
`maxcfengines`	Numero massimo di processi Cfengine concorrenti.
`repchar`	Carattere usato in sostituzione di '`/`' nei nomi di file.
`split`	Carattere usato per separare gli elenchi nelle variabili.
`sysadm`	Amministratore (nome o indirizzo di posta elettronica).
`checksumdatabase`	File destinato alla raccolta dei codici di controllo.

In generale, i nomi delle variabili sono distinti anche in base all'uso di maiuscole e minuscole; tuttavia, le variabili predefinite possono essere usate con qualunque combinazione di lettere maiuscole e minuscole.

Esiste anche un altro gruppo di variabili speciali, in sola lettura, definite per facilitare l'inserimento di caratteri speciali all'interno di stringhe, quando non è possibile fare altrimenti. Queste variabili sono elencate nella tabella u56.22.

Tabella u56.22. Variabili interne per la rappresentazione di caratteri speciali.

Variabile	Descrizione
`cr`	Ritorno a carrello: <*CR*>.
`dblquote`	Apici doppi: '`"`'.
`dollar`	Dollaro: '`$`'.
`lf`	Avanzamento di riga: <*LF*>.
`n`	Codice di interruzione di riga secondo l'architettura.
`quote`	Apice singolo: '`'`'.
`space`	Spazio singolo: <*SP*>.
`tab`	Tabulazione: <*TAB*>.

Le stringhe sono delimitate indifferentemente attraverso apici doppi e singoli, potendo usare anche gli apici singoli inversi. In pratica, si possono usare le forme seguenti:

```
"stringa"
'stringa'
`stringa`
```

Il significato è lo stesso e l'espansione delle variabili avviene in tutti i casi nello stesso modo. Disponendo di diversi tipi di delimitatori, è più facile includere questi simboli nelle stringhe stesse. In questo senso va considerato il fatto che non esistono sequenze di escape; al massimo si possono usare le variabili predefinite per la rappresentazione di caratteri particolari.

Le stringhe sono utilizzabili solo in contesti particolari, precisamente la definizione di valori da assegnare a una variabile dichiarata nella sezione '**`control`**' e i comandi di shell nella sezione '**`shellcommands`**' (che non è ancora stata mostrata).

Elenchi

Le variabili possono essere intese come contenenti un elenco di sottostringhe. In questi casi, la loro espansione può richiedere una valutazione ulteriore. Tutto ha inizio dalla variabile interna '**`split`**', che normalmente contiene il carattere '`:`'. In questo senso, si osservi l'esempio seguente:

```
control:
    actionsequence = ( tidy )
    elenco = ( "primo:secondo:terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Assegnando alla variabile '**`elenco`**' la stringa '**`primo:secondo:terzo`**', si ottiene l'indicazione di un elenco di tre sottostringhe: '**`primo`**', '**`secondo`**' e '**`terzo`**'. A questo punto, la direttiva contenuta nella sezione '**`tidy`**', si traduce nella cancellazione dei file '`/var/tmp/primo`', '`/var/tmp/secondo`' e '`/var/tmp/terzo`'. Volendo cambiare il simbolo di separazione delle sottostringhe si agisce nella variabile '**`split`**', come si vede nell'esempio seguente, che ottiene lo stesso risultato.


```
control:
    actionsequence = ( tidy )
    split = ( " " )
    elenco = ( "primo secondo terzo" )

tidy:
    /var/tmp/$(elenco) pattern=* age=0
```

Naturalmente, si può ottenere l'espansione di variabili del genere solo nei contesti in cui questo può avere significato.

Espressioni regolari

In contesti ben determinati, si possono indicare delle espressioni regolari. Cfengine utilizza le espressioni regolari ERE secondo le convenzioni GNU. Sono disponibili gli operatori riassunti nella tabella u56.25.

Tabella u56.25. Elenco degli operatori delle espressioni regolari.

Operatore	Descrizione
`\`	Protegge il carattere seguente da un'interpretazione diversa da quella letterale.
`^`	Ancora dell'inizio di una stringa.
`.`	Corrisponde a un carattere qualunque.
`$`	Ancora della fine di una stringa.
`|`	Indica due possibilità alternative alla sua sinistra e alla sua destra.
`( )`	Definiscono un raggruppamento.
`[ ]`	Definiscono un'espressione tra parentesi quadre.
`[`*`xy...`*`]`	Un elenco di caratteri alternativi.
`[`*`x-y`*`]`	Un intervallo di caratteri alternativi.
`[^...]`	I caratteri che non appartengono all'insieme.
`x``*`	Nessuna o più volte ***x***. Equivalente a '*`x`*`{0,}`'.
`x``?`	Nessuna o al massimo una volta ***x***. Equivalente a '*`x`*`{0,1}`'.
`x``+`	Una o più volte ***x***. Equivalente a '*`x`*`{1,}`'.
`x``{`*`n`*`}`	Esattamente ***n*** volte ***x***.
`x``{`*`n`*`,}`	Almeno ***n*** volte ***x***.
`x``{`*`n`*`,`*`m`*`}`	Da ***n*** a ***m*** volte ***x***.
`\b`	La stringa nulla all'inizio o alla fine di una parola.
`\B`	La stringa nulla interna a una parola.
`\<`	La stringa nulla all'inizio di una parola.
`\>`	La stringa nulla alla fine di una parola.
`\w`	Un carattere di una parola, praticamente '`[[:alnum:]_]`'.
`\W`	L'opposto di '`\w`', praticamente '`[^[:alnum:]_]`'.

Le espressioni regolari GNU includono anche le classi di caratteri (nella forma '`[:`***nome***`:]`', come prescrive lo standard POSIX, mentre mancano i simboli di collazione e le classi di equivalenza..

[1] **CFengine** GNU GPL

Permessi e proprietà 341
Sezione «control» 342
Impostazione delle variabili 342
Direttiva «actionsequence» 342
Direttiva «addclasses» 343
Sezione «classes» o «groups» 343
Sezione «copy» 343
Opzione «dest» 344
Opzione «recurse» 344
Opzione «type» 344
Opzione «purge» 345
Opzione «server» 345
Sezione «directories» 345
Sezione «disable» 345
Opzione «rotate» 346
Sezione «files» 346
Opzione «checksum» 346
Opzione «recurse» 346
Sezione «links» 346
Opzione «type» 347
Opzione «recurse» 348
Sezione «processes» 348
Opzione «signal» 348
Opzione «restart» 348
Opzione «matches» 348
Opzione «action» 349
Sezione «shellcommands» 349
Sezione «tidy» 349
Opzione «pattern» 349
Opzione «recurse» 349
Opzione «age» 349
Opzione «type» 350
Opzione «rmdirs» 350

Una volta compresa a grandi linee l'impostazione della configurazione di Cfengine, bisogna entrare nell'analisi specifica di ogni sezione che si voglia prendere in considerazione, dal momento che ognuna può avere le sue caratteristiche e le sue direttive specifiche. In questo capitolo si descrivono solo alcune sezioni tipiche, in modo superficiale, allo scopo di consentire un utilizzo elementare di Cfengine.

Si osservi che in generale non conta l'ordine in cui sono indicate le sezioni e le direttive all'interno delle sezioni; inoltre, le direttive possono utilizzare più righe senza bisogno di simboli di continuazione.

Permessi e proprietà

In più sezioni differenti si usano delle direttive che contengono opzioni con lo stesso nome e con lo stesso significato. Si tratta in particolare di quelle opzioni che definiscono le caratteristiche dei permessi e delle proprietà di file e directory. È il caso di mostrare queste opzioni una volta sola per tutte:

```
mode=modalità
```

```
owner=utente_proprietario
```

```
group=gruppo_proprietario
```

La modalità è un numero ottale oppure una stringa di permessi. La stringa di permessi può essere espressa come avviene con il comando '**chmod**'. Si osservino gli esempi seguenti.

Opzione	Descrizione
`mode=0775`	Imposta la modalità 0775_8 in modo preciso.
`mode=u+rwx`	Assegna sicuramente all'utente proprietario i permessi di lettura, scrittura ed esecuzione (o attraversamento).
`mode=o-rwx`	Toglie agli utenti diversi dall'utente proprietario e dal gruppo proprietario qualunque permesso di accesso.
`user=root` `group=root`	Stabilisce che l'utente e il gruppo proprietario deve essere '**root**', ammesso che Cfengine stia funzionando con i privilegi necessari per poter modificare la proprietà di file e directory.

Sezione «control»

La sezione '**control**' è quella fondamentale di ogni configurazione di Cfengine, dal momento che è attraverso questa, assieme alla direttiva '**actionsequence**', che si stabilisce l'utilizzo e l'ordine delle altre sezioni. In generale, la sintassi specifica di questa sezione è la seguente:

```
control:
    [espressione_classe::]
        nome = ( valore... )
        ...
    ...
    ...
```

È essenziale che, nelle direttive di assegnamento tipiche di questa sezione, le parentesi tonde siano spaziate sia all'interno che all'esterno.

Impostazione delle variabili

In questa sezione si dichiarano le variabili e si impostano quelle predefinite che richiedono un intervento. L'esempio seguente definisce la variabile predefinita '**domain**':

```
control:
    ...
    domain = ( brot.dg )
    ...
```

Direttiva «actionsequence»

Al nome '**actionsequence**' viene assegnato l'elenco di nomi di sezioni e di altre azioni da eseguire, in base all'ordine in cui si trovano in questo elenco:

```
control:
    ...
    actionsequence = ( azione_1 azione_2... )
    ...
```

A livello di utilizzo elementare, si fa riferimento sempre solo a nomi di sezione, mentre sono previsti altri nomi che identificano azioni particolari che non fanno capo a una sezione.

Direttiva «addclasses»

La direttiva '**addclasses**' è utilizzata per creare delle classi fittizie aggiuntive. L'esempio seguente aggiunge le classi '**bianco**' e '**nero**':

```
control:
    ...
    addclasses = ( bianco nero )
    ...
```

Si possono aggiungere delle classi anche con l'opzione '**-D***nome*' e si possono eliminare delle classi con l'opzione '**-N***nome*'.

Sezione «classes» o «groups»

La sezione '**classes**', ovvero anche '**groups**', è un po' anomala nella logica di Cfengine, dal momento che non rappresenta un'azione vera e propria, ma la dichiarazione di un raggruppamento di classi. Intuitivamente si comprende che questa cosa dovrebbe essere compito della sezione '**control**'. In effetti, questa sezione viene presa in considerazione comunque e non va annotata nella direttiva '**actionsequence**' della sezione '**control**'. La sintassi della dichiarazione di una classe nell'ambito di questa sezione, può essere di tre tipi:

```
classes:|groups:
    ...
    gruppo_di_classi = ( classe_1 classe_2... )
    ...
```

```
classes:|groups:
    ...
    classe = ( +dominio_nis )
    ...
```

```
classes:|groups:
    ...
    classe = ( "comando_di_shell" )
    ...
```

Nel primo caso si crea una classe che riproduce la somma di quelle indicate tra parentesi; nel secondo si ha una classe che rappresenta l'insieme degli elaboratori appartenenti al dominio NIS indicato; nel terzo si ottiene una classe se il comando indicato (delimitato tra virgolette) termina con successo, ovvero restituisce *Vero*.

Sezione «copy»

La sezione '**copy**' serve a copiare file nell'ambito dello stesso file system, oppure tra elaboratori differenti, attraverso il demone '**cfd**'. La copia viene fatta preparando prima un file con estensione '`.cfnew`', che alla fine viene rinominato nel modo previsto. Questa accortezza serve nella copia tra elaboratori, per evitare il danneggiamento dei file nel caso di interruzione della comunicazione nella rete. Salvo diversa indicazione, quando viene rimpiazzato un file attraverso la copia, quello vecchio viene conservato temporaneamente aggiungendogli l'estensione '`.cfsaved`'.

```
copy:
    [espressione_classe::]
        origine dest=destinazione [altre_opzioni]
        ...
    ...
    ...
```

Il contenuto della sezione '**copy**', può essere ovviamente suddiviso in classi, se ciò è utile. Alla fine, le direttive che possono essere contenute sono di un tipo solo, dove la prima informazione indica il nome del file, o il modello di file da copiare, mentre il resto sono del-

le opzioni nella forma ‘***nome=valore***’. Le opzioni di queste direttive sono numerose; qui ne vengono descritte solo alcune.

Opzione «dest»

```
dest=destinazione
```

Con questa opzione si definisce la destinazione della copia. Deve trattarsi di un oggetto dello stesso tipo dell’origine: se l’origine è un file normale, la destinazione deve essere un file normale; se l’origine è un collegamento simbolico la destinazione si riferisce a un collegamento simbolico; se l’origine è una directory, la destinazione deve essere una directory, in cui vengono copiati tutti i file che si trovano in quella originale (senza riprodurre le sottodirectory eventuali).

```
copy:
    /etc/passwd
        dest=/home/tizio/users
```

L’esempio mostra una situazione molto semplice, dove si vuole copiare il file ‘`/etc/passwd`’ nel file ‘`/home/tizio/users`’, oppure si vuole mantenere aggiornata la copia.

Opzione «recurse»

```
recurse={n_livelli|inf}
```

La copia di una directory può avvenire anche ricorsivamente, attraverso le sue sottodirectory, specificando il livello di ricorsione con questa opzione. Si assegna un numero intero, oppure la parola chiave ‘**`inf`**’. Il numero rappresenta la quantità di livelli di ricorsione da considerare, mentre la parola ‘**`inf`**’ richiede espressamente una ricorsione infinita.

```
copy:
    /etc
        dest=/home/tizio/copia_etc
        recurse=1
```

L’esempio mostra la copia del contenuto della directory ‘`/etc/`’ nella directory ‘`/home/tizio/copia_etc/`’. Dal momento che la ricorsione è limitata a un solo livello, si ottengono solo i file e le sottodirectory vuote (nel senso che non viene copiato anche il loro contenuto).

Opzione «type»

```
type={ctime|mtime|checksum|sum|byte|binary}
```

L’opzione ‘**`type`**’ definisce in che modo Cfengine può determinare se il file va copiato o meno. Normalmente si fa riferimento alla data di «creazione», intesa come quella in cui vengono modificati i permessi o comunque viene cambiato inode, nel senso che solo se il file di origine ha una data più recente viene fatta la copia. Intuitivamente si comprende il senso delle parole chiave ‘**`ctime`**’ e ‘**`mtime`**’: la prima fa riferimento esplicito a questa data di creazione, mentre la seconda fa riferimento alla data di modifica. In alternativa, le parole chiave ‘**`checksum`**’ o ‘**`sum`**’ richiedono un controllo attraverso un codice di controllo (una firma MD5) per determinare se il file originale è diverso e se è richiesta la copia.

Si osservi che nella copia tra elaboratori distinti, è l’elaboratore di destinazione che genera la firma MD5 del suo file e la invia all’elaboratore di origine per il confronto. Pertanto è nell’elaboratore di origine che avviene la comparazione delle firme e in caso di diversità avviene la trasmissione del file di origine.

Le parole chiave ‘**`byte`**’ e ‘**`binary`**’ richiedono un confronto completo dei file byte per byte.

```
copy:
    /etc/passwd
        dest=/home/tizio/users
        type=checksum
```


L’esempio mostra il caso della copia del file ‘`/etc/passwd`’ nel file ‘`/home/tizio/users`’, verificando la necessità di aggiornare la copia attraverso un codice di controllo.

Opzione «purge»

```
purge={true|false}
```

L’opzione ‘**`purge`**’, se attivata assegnando la parola chiave ‘**`true`**’, abilita l’eliminazione dei file che nell’origine non sono più presenti.

```
copy:
    /etc
        dest=/home/tizio/copia_etc
        recurse=inf
        purge=true
```

L’esempio mostra la copia del contenuto della directory ‘`/etc/`’ nella directory ‘`/home/tizio/copia_etc/`’, con ricorsione infinita, specificando anche che nella destinazione devono essere eliminati i file e le directory che non sono più presenti nell’origine.

Opzione «server»

```
server=nodo_remoto
```

Questa opzione si usa quando si vuole copiare un file remoto; per la precisione, serve a specificare che l’origine si trova presso un altro elaboratore. Per riuscire a copiare attraverso elaboratori, è necessario che il nodo servente sia stato predisposto con il demone ‘**`cfd`**’; inoltre, è necessario specificare la variabile ‘**`domain`**’ nella sezione di controllo (‘**`control`**’).

Sezione «directories»

```
directories:
    [espressione_classe::]
        directory [opzioni]
        ...
    ...
    ...
```

Le direttive della sezione ‘**`directories`**’ servono a richiedere la presenza di directory determinate, specificando eventualmente le caratteristiche necessarie. Se le directory in questione mancano, vengono create; se le caratteristiche non corrispondono, queste vengono modificate.

Le opzioni più importanti sono quelle che definiscono i permessi e la proprietà, come descritto nella sezione u0.1.

```
directories:
    /           mode=0755 owner=root group=root
    /etc        mode=0755 owner=root group=root
    /bin        mode=0755 owner=root group=root
    /dev        mode=0755 owner=root group=root
    /sbin       mode=0755 owner=root group=root
    /lib        mode=0755 owner=root group=root
    /usr        mode=0755 owner=root group=root
    /tmp        mode=1777 owner=root group=root
```

L’esempio mostra una serie di direttive della sezione ‘**`directories`**’ con lo scopo di salvaguardare la presenza, i permessi e la proprietà di alcune directory importanti.

Sezione «disable»

```
disable:
    [espressione_classe::]
        file [opzioni]
        ...
    ...
    ...
```

La sezione ‘**`disable`**’ serve a elencare un gruppo di file che si vogliono «disabilitare». L’idea è che questi file non devono essere presenti, ma non si vogliono nemmeno cancellare. In pratica, se vengono trovati, si aggiunge loro l’estensione ‘`.cfdisabled`’.

```
disable:
    /etc/hosts.equiv
```

L'esempio mostra una situazione tipica, in cui si vuole evitare che esista il file '`/etc/hosts.equiv`', pur lasciando la possibilità di verificare il suo contenuto originale.

Opzione «rotate»

```
rotate={n|empty}
```

Eccezionalmente, se si utilizza l'opzione '**`rotate`**', si fa riferimento implicitamente a file di registrazioni (*log*), che conviene spezzare periodicamente. Assegnando un numero intero all'opzione, si specifica la quantità di livelli da conservare. Per esempio, assegnando il valore '**`2`**', si fa in modo che il file venga rinominato aggiungendo l'estensione '`.1`', mentre un eventuale file preesistente con lo stesso nome verrebbe rinominato sostituendo l'estensione '`.1`' con '`.2`'.

Se si assegna la parola chiave '**`empty`**', non si salvano le versioni precedenti, annullando semplicemente il contenuto del file.

```
disable:
    /var/log/wtmp rotate=7
```

L'esempio mostra la richiesta di mettere da parte il file '`/var/log/wtmp`', in modo da ricominciare con un file vuoto, mantenendo sette copie precedenti, da '`/var/log/wtmp.1`' a '`/var/log/wtmp.7`'.

Sezione «files»

```
files:
    [espressione_classe::]
        file [opzioni]
        ...
    ...
    ...
```

Le direttive della sezione '**`files`**' servono a richiedere la presenza di file determinati, specificando eventualmente le caratteristiche necessarie. Se i file in questione mancano, vengono creati (vuoti); se le caratteristiche non corrispondono, queste vengono modificate per quanto possibile.

Le opzioni più importanti sono quelle che definiscono i permessi e la proprietà, come descritto nella sezione u0.1. Tuttavia è importante anche la possibilità di controllare che i file in questione non siano stati modificati.

Opzione «checksum»

```
checksum=md5
```

L'opzione '**`checksum`**' (a cui può essere assegnato solo il valore '**`md5`**') consente di richiedere la verifica dei file attraverso un codice di controllo. Inizialmente, questo codice di controllo deve essere accumulato da qualche parte e precisamente si tratta del file dichiarato nella variabile '**`checksumdatabase`**' nella sezione di controllo ('**`control`**').

Opzione «recurse»

```
recurse={n_livelli|inf}
```

Anche se si sta facendo riferimento principalmente a file, è consentito indicare directory intere, specificando il livello di ricorsione attraverso l'opzione '**`recurse`**', già descritta in precedenza.

La sezione '**`files`**' è orientata ai file. Tuttavia, se si richiede l'impostazione di permessi specifici, questi potrebbero interferire con quelli delle directory, nel momento in cui si fa riferimento a queste. Per risolvere il problema, Cfengine aggiunge i permessi di attraversamento necessari alle directory.

Sezione «links»

```
links:
    [espressione_classe::]
        collegamento {-|+}>[!] oggetto_originale [opzioni]
        ...
    ...
    ...
```

La sezione '**`links`**' serve per creare, aggiornare e sistemare dei collegamenti simbolici. In generale si distingue tra collegamenti singoli o collegamenti multipli. La differenza sta nell'uso dell'operatore '**`->`**' oppure '**`+>`**'.

I collegamenti singoli riguardano un solo collegamento simbolico. Se il collegamento esiste già, viene verificato che corrisponda a quanto descritto nella direttiva, altrimenti si ottiene una segnalazione di errore.

```
/usr/local      -> /mia_dir/usr/local
```

L'esempio mostra una direttiva in cui si vuole che sia creato e mantenuto il collegamento simbolico '`/usr/local`', che deve puntare alla directory reale '`/mia_dir/usr/local/`'.

Se il collegamento simbolico esiste già ma non corrisponde, oppure si tratta di un file, si può imporre la sua correzione con l'aggiunta del punto esclamativo:

```
/usr/local      ->! /mia_dir/usr/local
```

In tal caso, se '`/usr/local`' fosse un file, il suo nome verrebbe modificato in '`/usr/local.cfsaved`' e il collegamento potrebbe così essere sistemato.

I collegamenti multipli si fanno indicando una directory di destinazione e una directory di origine, come nell'esempio seguente:

```
/usr/local/bin  +> /mia_dir/usr/local/bin
```

In questi casi si vogliono generare nella directory '`/usr/local/bin/`' tanti collegamenti simbolici quanti sono i file nella directory '`/mia_dir/usr/local/bin/`'.

Anche nel caso di collegamenti multipli si può usare il punto esclamativo per richiedere la correzione necessaria al completamento dell'operazione.

In generale, non vengono eliminati i collegamenti riferiti a file o directory non più esistenti. Per ottenere questo risultato, che potrebbe essere particolarmente utile in presenza di collegamenti multipli, occorre usare l'opzione '**`-L`**' nella riga di comando dell'eseguibile '**`cfengine`**'.

La creazione di un collegamento simbolico può richiedere la creazione delle directory che lo precedono. In condizioni normali ciò avviene automaticamente, senza bisogno di preoccuparsene.

Opzione «type»

```
type={hard|relative|absolute}
```

La sezione '**`links`**' è pensata fondamentalmente per gestire collegamenti simbolici. Tuttavia, con questa opzione è possibile richiedere la creazione di collegamenti fisici, oltre alla possibilità di specificare il tipo di collegamento simbolico che si vuole ottenere: assoluto o relativo.

Indipendentemente dalle possibilità del sistema operativo, Cfengine non può creare dei collegamenti fisici che puntano a directory.

Opzione «recurse»

```
recurse={n_livelli|inf}
```

Dal momento che è consentita la generazione di collegamenti multipli, diventa opportuna la possibilità di specificare il livello di ricorsione attraverso l'opzione '**recurse**', già descritta in precedenza.

Sezione «processes»

```
processes:
    [espressione_classe::]
        "espressione_regolare" [opzioni]
        ...
    ...
    ...
```

La sezione '**processes**' serve a individuare dei processi in funzione, attraverso un'analisi di quanto restituito dal comando '**ps**' del sistema operativo. Lo scopo può essere di due tipi: inviare un segnale al processo o ai processi individuati, oppure eseguire un comando per riavviarli se questi risultano mancanti.

Si deve osservare che ogni direttiva individua uno o più processi in base a un'espressione regolare, delimitata tra virgolette. In tal modo si può fare riferimento a tutto ciò che appare nel rapporto generato dal comando '**ps**', non soltanto il nome del processo. Tuttavia, ciò significa anche che occorre predisporre bene queste espressioni regolari, per non incorrere in errori.

Opzione «signal»

```
signal=nome_del_segnale
```

attraverso l'opzione '**signal**' si richiede espressamente l'invio del segnale specificato. In mancanza di questa, non si invia alcun segnale. Il nome da assegnare dipende dal sistema operativo utilizzato, anche se in generale si tratta di nomi abbastanza standardizzati.

```
processes:
    "inetd" signal=hup
```

L'esempio mostra il caso in cui si cerchi il processo individuato dalla stringa '**inetd**', per inviargli il segnale SIGHUP.

Opzione «restart»

```
restart "comando_di_shell"
```

Se non si trova una corrispondenza con l'espressione regolare indicata, si può ottenere l'avvio di un comando che presumibilmente serve ad avviare il processo relativo. Per questo si utilizza l'opzione '**restart**' che, come si vede dal modello sintattico, **non utilizza** il simbolo '**=**' per l'assegnamento.

Opzione «matches»

```
matches=[>|<]n
```

È possibile individuare una quantità di processi che corrispondono all'espressione regolare di partenza, definendo la quantità attesa. Se si usano gli operatori '**<**' e '**>**', ci si aspettano più di ***n*** processi, o meno di ***n*** processi, perché la condizione si avveri. Diversamente si attendono esattamente ***n*** processi.

Di solito, questa opzione si abbina soltanto alla richiesta di un avvertimento, attraverso l'opzione '**action=warn**'.

```
processes:
    "telnetd" matches=<7 action=warn
```

Questo esempio mostra il caso in cui si voglia essere avvisati se si trovano sette o più processi corrispondenti alla stringa '**telnetd**'.

Può sembrare strana l'interpretazione dei simboli '**>**' e '**<**'. In realtà si deve vedre la cosa dal lato opposto: con '**>**' ci si aspetta che i processi siano meno della quantità indicata, perché non debba essere eseguita l'azione; nello stesso modo, con '**<**' ci si aspetta che i processi siano di più di quanto indicato perché l'azione non sia eseguita.

Opzione «action»

```
action={signal|do|warn}
```

L'opzione '**action**' stabilisce il da farsi, quando si verificano le condizioni richieste per intervenire. Le parole chiave '**signal**' o '**do**' richiedono espressamente l'invio del segnale stabilito con l'opzione '**signal**'; la parola chiave '**warn**' richiede solo una segnalazione di avvertimento.

Sezione «shellcommands»

```
shellcommands:
    [espressione_classe::]
        "comando" [opzioni]
        ...
    ...
    ...
```

La sezione '**shellcommands**' serve a inserire dei comandi di shell, debitamente delimitati tra virgolette. Di solito, non si utilizzano le opzioni, anche se in situazioni particolari possono essere utili.

Sezione «tidy»

```
tidy:
    [espressione_classe::]
        directory pattern=modello [altre_opzioni]
        ...
    ...
    ...
```

La sezione '**tidy**' è fatta per eliminare dei file non desiderati. Come si può osservare dal modello sintattico, le direttive iniziano dalla definizione di una directory di partenza, per cui diventa necessario specificare i file da eliminare attraverso un modello con l'opzione '**pattern**'.

Opzione «pattern»

```
pattern=modello
```

Attraverso l'opzione '**pattern**' si specifica il file o i file da prendere in considerazione nella directory di partenza. Il modello si può realizzare utilizzando i soliti simboli speciali, '*****' e '**?**', con il significato consueto: qualunque stringa, oppure un solo carattere.

Opzione «recurse»

```
recurse={n_livelli|inf}
```

È consentita la cancellazione di file anche attraverso le sottodirectory, utilizzando l'opzione '**recurse**', come già è stato mostrato in precedenza.

Opzione «age»

```
age=n_giorni
```

L'opzione '**age**' consente di specificare quanto tempo devono avere i file per poter essere cancellati. Se il tempo è stato raggiunto

o superato, si ottiene la cancellazione. Questo tempo si riferisce in modo predefinito alla data di accesso, ma può essere cambiato con l'opzione '**type**'.

```
tidy:
    /tmp      pattern=*  recurse=inf  age=1
    /var/tmp  pattern=*  recurse=inf  age=7
```

L'esempio mostra un caso molto semplice, in cui si vuole ripulire il contenuto delle directory '`/tmp/`' e '`/var/tmp/`', per i file che sono vecchi rispettivamente un giorno e sette giorni.

Opzione «type»

```
type=[ctime|mtime|atime]
```

La vecchiaia di un file può essere valutata in base alla data di «creazione», intesa come cambiamento di inode, di modifica o di accesso, assegnando rispettivamente le parole chiave '**ctime**', '**mtime**' o '**atime**' all'opzione '**type**'. Questo serve per stabilire il modo corretto di interpretazione del valore assegnato all'opzione '**age**'.

Opzione «rmdirs»

```
rmdirs=[true|false|all|sub]
```

In condizioni normali, non si ottiene la cancellazione delle directory. Per questo, occorre usare l'opzione '**rmdirs**', a cui si assegnano le parole chiave che si vedono nel modello sintattico. In condizioni normali, è come se fosse assegnata la parola chiave '**false**', che impedisce la cancellazione. Se si richiede la cancellazione, si elimina anche la directory di partenza, corrispondente al modello richiesto. Al contrario, assegnando la parola chiave '**sub**', si preserva la directory di partenza.

Cfengine attraverso la rete

Configurazione e avvio del demone 351
Configurazione essenziale di «cfd.conf» 351
Filosofia del sistema di distribuzione di Cfengine 353

Cfengine consente anche un utilizzo attraverso la rete, per mezzo del demone '**cfd**', che viene avviato nell'elaboratore che offre il proprio servizio.

L'utilizzo più semplice di questa possibilità di Cfengine sta nella copia di file attraverso la rete, per sincronizzare gli elaboratori clienti. Per la precisione, questo particolare è l'unica cosa che viene mostrata qui, in questo capitolo.

Configurazione e avvio del demone

Il servizio relativo al demone '**cfd**' prevede l'accesso alla porta TCP 5308, che pertanto non è privilegiata e consente l'avvio del demone anche senza i privilegi dell'utente '**root**', se questo può essere utile per qualche motivo. Nel file '`/etc/services`' dovrebbe esserci pertanto una riga simile a quella seguente:

```
cfengine        5308/tcp
```

Per funzionare, il demone '**cfd**' richiede la presenza del file '`cfd.conf`', nella directory corrente nel momento dell'avvio del demone, che ha una struttura simile a quella di '`cfengine.conf`'.

Oltre a questo file essenziale, occorre tenere presente che il demone tiene in considerazione anche il contenuto dei file '`/etc/hosts.allow`' e '`/etc/hosts.deny`', per controllare gli accessi.

Una volta predisposto il sistema di configurazione, basta avviare il demone '**cfd**', con i privilegi dell'utente '**root**', se necessario, oppure con i privilegi di un utente comune.

```
# cfd [Invio]
```

Alcune opzioni del demone '**cfd**' sono molto utili per consentire l'analisi del file di configurazione e per poter tenere sotto controllo ciò che avviene effettivamente durante la connessione. Queste opzioni sono riepilogate nella tabella u58.2.

Tabella u58.2. Elenco delle opzioni essenziali di '**cfd**'.

Opzione	Descrizione
`-h` `--help`	Elenca brevemente le opzioni disponibili.
`-d` `--debug`	Rimane in primo piano e mostra ciò che accade.
`-v` `--verbose`	Mostra informazioni dettagliate.
`-p` `--parse-only`	Si limita a scandire il file di configurazione.

Può essere interessante il controllo della configurazione attraverso l'opzione '**-p**', unita opportunamente all'opzione '**-v**'. Inoltre, per verificare le connessioni, soprattutto alla ricerca delle motivazioni per cui qualcosa non funziona come si vorrebbe, conviene utilizzare l'opzione '**-d**', sempre in combinazione con '**-v**'.

```
# cfd -d -v [Invio]
```

Configurazione essenziale di «cfd.conf»

Il file '`cfd.conf`' ha una vaga somiglianza con il file di configurazione di un cliente normale di Cfengine. In particolare, ci sono le sezioni e possono essere presenti le classi, solo che hanno va-

lore esclusivamente nei confronti dell'elaboratore in cui si trova a funzionare il demone.

Generalmente, è probabile che non si faccia uso di classi in un file '`cfg.conf`' e qui non si mostrano esempi in tal senso. La sintassi semplificata ed essenziale di questo file, viene mostrata dal modello seguente. Si tenga presente che non vengono mostrate tutte le direttive, ma solo quelle che devono essere conosciute necessariamente.

```
control:
    [domain = ( dominio )]
    [maxconnections = ( numero_massimo_di_connessioni_indipendenti )
]
    [allowconnectionsfrom = ( numero_ip [numero_ip]… )]
    [denyconnectionsfrom = ( numero_ip [numero_ip]… )]
    [allowmultipleconnectionsfrom = ( numero_ip [numero_ip]
… )]
    [logallconnections = ( true|false )]

admit:|grant:
    file_o_directory            nodi_indicati_con_caratteri_jolly
    …

deny:
    file_o_directory            nodi_indicati_con_caratteri_jolly
    …
```

Si può osservare la presenza di una sezione di controllo, simile a quella dei clienti Cfengine. Questa sezione può anche risultare vuota.

Le sezioni '**`admit`**' (o '**`grant`**') e '**`deny`**', permettono di stabilire l'accessibilità di file e di directory, a degli elaboratori identificati per nome, anche in modo parziale attraverso caratteri jolly. Si intende che la sezione '**`admit`**' o '**`grant`**' serva a elencare i file e le directory accessibili, mentre la sezione '**`deny`**' serve a escludere successivamente parte di quanto precedentemente concesso.

Nella sezione di controllo, le direttive '**`maxconnections`**', '**`allowconnectionsfrom`**', '**`denyconnectionsfrom`**' e '**`allowmultipleconnectionsfrom`**', limitano o concedono gli accessi attraverso l'indicazione di un elenco di indirizzi IP. In generale, questo può essere un mezzo ulteriore di controllo di sicurezza per gli accessi, dal momento che spesso è sufficiente l'uso delle sezioni '**`admit`**' e '**`deny`**'. In particolare, ogni cliente Cfengine che accede, ha la possibilità di aprire una sola connessione, mentre con la direttiva '**`allowmultipleconnectionsfrom`**' è possibile autorizzare un accesso multiplo agli indirizzi indicati.

L'uso delle altre direttive indicato dovrebbe essere intuitivo; inoltre, nella sezione di controllo è possibile dichiarare delle variabili, nello stesso modo della configurazione dei clienti Cfengine.

È importante ricordare che i percorsi di cui si concede l'accesso, devono essere reali, perché i collegamenti simbolici non vengono presi in considerazione. Questo tipo di errore lo si può individuare utilizzando l'opzione '**`-d`**' quando si avvia '**`cfd`**'.

A titolo di esempio viene mostrato un caso molto semplice di configurazione, in cui si concede l'accesso alle directory '`/usr/local/file_pubblici1/`' e '`/usr/local/file_pubblici2/`', creando appositamente due variabili per semplificarne l'indicazione; inoltre si concede l'accesso anche ai file '`/etc/passwd`' e '`/etc/group`'. Per la precisione, la directory '`/usr/local/file_pubblici1/`' risulta accessibile a tutti, mentre '`/usr/local/file_pubblici2/`' è accessibile solo ai domini **.brot.dg* e **.mehl.dg*; inoltre, i due file '`/etc/passwd`' e '`/etc/group`' sono accessibili esclusivamente dal dominio **.brot.dg*. Infine, per qualche motivo, si esclude l'accesso alla directory '`/usr/local/file_pubblici2/particolare/`' al dominio **.mehl.dg*. Ogni cliente può aprire una sola connessione e sono consentiti un massimo di 10 accessi simultanei.

```
control:
    pubblici1 = ( /usr/local/file_pubblici1 )
    pubblici2 = ( /usr/local/file_pubblici2 )
    maxconnections = ( 10 )

admit:
    $(pubblici1) *
    $(pubblici2) *.brot.dg *.mehl.dg
    /etc/passwd  *.brot.dg
    /etc/group   *.brot.dg

deny:
    $(pubblici2)/particolare *.mehl.dg
```

Filosofia del sistema di distribuzione di Cfengine

È il caso di osservare che, contrariamente a Rsync, il cliente Cfengine contatta il servente per ottenere qualcosa e non per inviare lì un file.

Quando la trasmissione di un file è sottoposta al confronto di un codice di controllo, è il cliente Cfengine che invia il suo codice di controllo al servente, il quale verifica la necessità o meno di trasmettere il file aggiornato.

Parte xxi

Connettività con sistemi Dos

Dos IPv4 357

Driver di pacchetto 357
Libreria WATTCP 359
Applicazioni standard per WATTCP 359
ABC-nslookup 360
MiniTelnet 360
SSHDOS 361
Bobcat (Lynx) 362
PPRD: servente di rete per la stampa 362
Trout 363
Talk 363
DosLynx 364
NCSA Telnet 365
POPMail 366
PCroute 368
Riferimenti 370

Dos PPP 371

Composizione 371
Configurazione e script 371
Connessione in pratica 372

Introduzione a NOS-KA9Q -- IPv4 per Dos 375

Preparazione 376
Interfacce, instradamento e nomi 378
Gestione delle sessioni 381
Attività nel sistema locale 381
Gestione della rete e delle connessioni 382
NOS come cliente 383
NOS come servente 384
NOS come router IPv4 385

nanoDos 387

Organizzazione della distribuzione 387
Installazione 387
Configurazione 388
Utilizzo 389
Conclusione 389
Riferimenti 389

Driver di pacchetto 357
Libreria WATTCP 359
Applicazioni standard per WATTCP 359
ABC-nslookup 360
MiniTelnet 360
SSHDOS 361
Bobcat (Lynx) 362
PPRD: servente di rete per la stampa 362
Trout 363
Talk 363
DosLynx 364
NCSA Telnet 365
File «CONFIG.TEL» 365
File «KEYFILE.TEL» 365
File «PASSWORD.TEL» 366
File «SERVICES.TEL» 366
File «TELBIN.EXE» 366
File «FTPBIN.EXE» 366
File «FINGER.EXE» 366
POPMail 366
Menù «Setup» 367
Menù «Window» 367
Menù «=» 368
PCroute 368
Configurazione dei driver di pacchetto 368
Configurazione di PCroute 368
Conclusione 370
Riferimenti 370

Come sistemi operativi liberi, i sistemi GNU costituiscono la scelta ottimale, se non altro dal punto di vista economico, per la realizzazione di reti locali. Ma anche il recupero di vecchia tecnologia può essere di grande aiuto: il vecchio hardware basato su i286 può essere introdotto in una rete TCP/IP per servizi classici quali TELNET, FTP e altro.

Negli esempi che appaiono nelle sezioni che seguono si immagina di avere una piccola rete locale con due elaboratori.

1. 192.168.1.1 *dinkel.brot.dg*, sistema GNU, con funzionalità di router e di servente DNS;
2. 192.168.1.15 *dos.brot.dg* con il sistema Dos.

Il secondo elaboratore è quello che si vuole utilizzare con i programmi Dos descritti in questo capitolo. Per quanto riguarda il caso particolare del programma PCroute, vengono mostrati esempi con dati differenti.

Prima di proseguire, è importante evitare di farsi illusioni: si tratta di programmi molto deboli, utili solo per IPv4 e a volte incapaci di attraversare i router.

Driver di pacchetto

Per poter comunicare attraverso un elaboratore con sistema operativo Dos in una rete TCP/IP occorre un cosiddetto ***driver di pacchetto*** (*packet driver*), ovvero un driver software in grado di fornire un minimo servizio basato sul protocollo IP. La raccolta di driver di pacchetto più comune è quella della Crynwr [1] (*http://www.crynwr.com*).

I programmi che si intendono utilizzare devono essere predisposti per il tipo di driver di pacchetto a disposizione.

Una raccolta di driver di pacchetto organizzata da Crynwr, può essere ottenuta presso Simtel.Net (la nota distribuzione di software *shareware* e *freeware* per Dos e MS-Windows) all'indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/pktdrvr/pktd11.zip* .

All'interno della raccolta si può trovare un lungo elenco di driver per vari modelli di schede di rete Ethernet. In particolare, vale la pena di soffermarsi sui driver per le schede Ethernet NE2000 e per la connessione PLIP attraverso la porta parallela.

- '**`NE2000.COM`**'

  Si tratta del driver adatto per la connessione attraverso schede Ethernet compatibili NE2000. Questo programma deve essere avviato con i parametri necessari per poter comunicare con la scheda di rete e con le applicazioni.

  `NE2000.COM` [*opzioni*] [*irq_software*] [*irq_della_scheda*] [*i/o_della_scheda*]

  Le schede NE2000 vengono configurate, attraverso ponticelli o software di configurazione, predisponendo un IRQ e un indirizzo di I/O. Oltre a queste indicazioni, è necessario specificare un indirizzo IRQ aggiuntivo che viene utilizzato per la comunicazione tra i programmi e il driver stesso. La scelta di questo IRQ software è la parte più delicata. L'indirizzo $7E_{16}$ dovrebbe andare bene. Supponendo di avere installato una scheda configurata con IRQ 11 ($0B_{16}$) e indirizzo di I/O 300_{16}, si deve avviare il driver nel modo seguente:

  ```
  NE2000.COM 0x7e 0x0b 0x300
  ```

  È opportuno aggiungere questa riga all'interno del file '`AUTOEXEC.BAT`'.

- '**`PLIP.COM`**'

  Si tratta del driver adatto per la connessione attraverso la porta parallela con un cavo PLIP. Questo programma deve essere avviato con i parametri necessari per poter comunicare con la scheda di rete e con le applicazioni.

  `PLIP.COM` [*opzioni*] [*irq_software*] [*irq_della_porta*] [*i/o_della_porta*]

  Come nel caso delle schede NE2000, è necessario specificare un indirizzo IRQ aggiuntivo che viene utilizzato per la comunicazione tra i programmi e il driver stesso. L'indirizzo $7E_{16}$ dovrebbe andare bene. Supponendo di avere a disposizione una porta parallela che utilizza IRQ 7 (07_{16}) e indirizzo di I/O 378_{16}, si deve avviare il driver nel modo seguente:

  ```
  PLIP.COM 0x7e 0x07 0x378
  ```

  È opportuno aggiungere questa riga all'interno del file '`AUTOEXEC.BAT`'.

I driver di pacchetto Crynwr e altri simili, possono essere rimossi dalla memoria residente attraverso un programma speciale che accompagna la raccolta stessa. Si tratta di '**`TERMIN.COM`**' che richiede soltanto l'indicazione dell'indirizzo IRQ software con il quale è stato installato il driver:

`TERMIN.COM` *irq_software*

Per esempio, per eliminare il driver installato utilizzando l'indirizzo $7E_{16}$, viene eliminato dalla memoria residente con il comando seguente:


```
TERMIN.COM 0x7e
```

Libreria WATTCP

software non libero: non è consentita la distribuzione di versioni modificate e non è consentita la commercializzazione

WATTCP [2] (*University of Waterlo TCP*) è una libreria utilizzata da alcuni programmi per accedere alle funzionalità TCP/IP. Di conseguenza, questi programmi hanno in comune lo stesso tipo di file di configurazione, che normalmente è denominato '`WATTCP.CFG`'.

Generalmente, questi programmi incorporano completamente il codice della libreria WATTCP, pertanto, i programmi sono autonomi, ma possono usare in comune lo stesso file di configurazione.

Questo file si compone di direttive molto semplici, in cui si assegna idealmente un valore a una variabile:

variabile_di_configurazione **`=`** ***valore***

In generale, viene definito l'indirizzo IP, il nome corrispondente e la maschera di rete (o della sottorete), come si vede nell'esempio seguente:

```
HOSTNAME = dos
MY_IP = 192.168.1.15
NETMASK = 255.255.255.0
```

Inoltre, di solito si indicano anche i serventi DNS,[3] il nome del proprio dominio e il router (gateway):

```
GATEWAY = 192.168.1.1
NAMESERVER = 192.168.1.1
DOMAINSLIST = brot.dg
```

Può essere interessante anche la definizione della dimensione massima dei pacchetti (MSS, *Max segment size*), se per qualche motivo il driver di pacchetto dovesse avere delle limitazioni:

```
MSS = 512
```

In breve, un esempio completo, senza l'indicazione della dimensione massima dei pacchetti:

```
MY_IP = 192.168.1.15
NETMASK = 255.255.255.0
GATEWAY = 192.168.1.1
NAMESERVER = 192.168.1.1
HOSTNAME = dos
DOMAINSLIST = brot.dg
```

Alcuni programmi potrebbero richiedere la presenza di una variabile di ambiente che permetta loro di individuare facilmente la collocazione e il nome del file di configurazione. Per esempio, se si tratta del file '`C:\TCPIP\WATTCP.CFG`', potrebbe essere richiesto di includere nel file '`AUTOEXEC.BAT`' la riga seguente:

```
SET WATTCP.CFG=\TCPIP
```

I programmi che utilizzano questa libreria, fanno spesso riferimento a file standard dei sistemi Unix, che devono trovare nella directory corrente:

'`HOSTS`'	contiene un elenco di indirizzi IP associati al nome a dominio corrispondente, equivalente al noto '`/etc/hosts`' dei sistemi Unix
'`PROTOCOL`'	contiene l'elenco dei protocolli con i nomi associati, equivalente al noto '`/etc/protocols`' dei sistemi Unix
'`SERVICES`'	contiene l'elenco dei servizi di rete, equivalente al noto '`/etc/services`' dei sistemi Unix.

Applicazioni standard per WATTCP

Assieme alla libreria WATTCP, si trovano i sorgenti di alcuni programmi comuni, [4] diffusi in forma binaria in un archivio compresso denominato '`apps.zip`'. La tabella seguente ne elenca alcuni:

`tcpinfo`	mostra le informazioni tratte dal file di configurazione '`wattcp.cfg`', consentendone il controllo;
`ping` ***nodo***	esegue una richiesta di eco ICMP verso il nodo indicato;
`finger` [***utente@***]***nodo***	richiede le informazioni disponibili sugli utenti del nodo indicato;
`rexec` ***nodo*** [***utente*** [***parola_d'ordine***]] ***comando***	esegue un comando remoto attraverso il protocollo RSH;
`lpr` [***coda***] ***nodo file***	invia un file alla stampante remota;
`lpq -P`***coda*** `-S`***nodo***	interroga la coda di una stampante remota;
`ftp` [***utente@***]***nodo***	richiede l'instaurazione di un collegamento FTP con il nodo indicato;

ABC-nslookup

ABC-nslookup [5] consente l'interrogazione di un servizio DNS. Si tratta di un programma molto semplice che utilizza la libreria WATTCP e anche la libreria ABC, ma la seconda richiede la preparazione di altri file; in particolare, per ciò che riguarda ABC-nslookup, è necessario predisporre un file contenente l'elenco dei serventi DNS a disposizione.

Il pacchetto originale del programma, corrispondente al file '`nslb01a.zip`', include una sottodirectory che dovrebbe essere riprodotta tale e quale nella radice del disco: '`\ETC\`'. Questa directory contiene in particolare il file '`RESOLV.CNF`', che corrisponde in pratica al file '`/etc/resolv.conf`' dei sistemi Unix. Eventualmente, se si desidera collocare questi file in una posizione diversa, basta definire la variabile di ambiente '**`ABCETCDIR`**'; per esempio, si può scrivere nel file '`AUTOEXEC.BAT`':

```
SET ABCETCDIR=\TCPIP
```

In questo caso, si intende dire che i file di tale directory si trovano invece in '`\TCPIP\`'.

È la libreria ABC che richiede la presenza di alcuni fine nella directory '`\ETC\`', pertanto è questo il motivo del nome della variabile di ambiente.

Il pacchetto si compone di due eseguibili, che cercano il file '`WATTCP.CFG`' nella directory corrente, ignorando la variabile di ambiente '**`WATTCP.CFG`**':

`NSLOOKUP` [***nodo_da_trovare*** [***servente_dns***]]

`NSQUERY` [`-d`] ***nodo_da_trovare*** [***servente_dns***]

I due comandi consentono di interrogare un servente DNS per risolvere un nome in numero e viceversa, oppure per avere maggiori dettagli sulle registrazioni del DNS che riguardano il nodo cercato. Se non viene indicato il servente DNS nella riga di comando, si fa riferimento a quanto indicato nella configurazione, precisamente nel file '`RESOLV.CNF`', che a questo proposito si compila come quello dei sistemi Unix.

MiniTelnet

MiniTelnet [6] è un piccolo programma cliente per il protocollo TELNET, basato sulla libreria WATTCP. Il programma eseguibile corrispondente è '**`MT.EXE`**', che si usa così:

`MT` ***nodo*** [`-P`***porta***] [`-E`***emulazione***] [`-K`***tastiera***]

L'emulazione del terminale viene definita con l'opzione '**`-E`**', secondo la tabella:

Opzione	Descrizione
`-EVT52`	emula un terminale di tipo VT52;
`-EHeath19`	emula una variante del terminale VT52;
`-EVT102`	emula un terminale VT102;
`-EVT200`	emula un terminale VT200;
`-EANSI`	emula un terminale ANSI;

Per quanto riguarda l'emulazione della tastiera, il pacchetto di MiniTelnet include alcuni file di esempio, con estensione '`.KBD`'. Per selezionare uno di questi file, si utilizza l'opzione '**`-K`**', seguita dalla radice del nome di questi file. Per esempio, per utilizzare il file '`VT-AT.KBD`', si deve usare l'opzione '**`-KVT-AT`**'.

Per accedere a un sistema GNU con MiniTelnet, attraverso un vecchio elaboratore con tastiera standard, può essere conveniente l'uso della sintassi seguente:

`MT` ***nodo*** `-EANSI -KVT-AT`

In questo modo si seleziona l'emulazione ANSI e il file '`VT-AT.KBD`' per la tastiera. Tuttavia, il servente TELNET potrebbe non essere in grado di passare l'informazione sul tipo di terminale utilizzato alla shell, pertanto conviene impostare manualmente la variabile '**`TERM`**' una volta iniziato il collegamento:

`$` **`export TERM=linux`** [*Invio*]

L'esempio riguarda il caso di un sistema GNU/Linux, dove la voce '**`linux`**' per identificare il tipo di terminale sembra essere la più vicina al funzionamento ottimale.

SSHDOS

SSHDOS [7] è un cliente per il protocollo SSH nelle versioni 1.∗. Si tratta di un programma che usa la libreria WATTCP, pertanto non crea problemi di configurazione. Il vero problema, semmai, riguarda la versione del protocollo, dal momento che un servente che offre solo il protocollo 2, non può comunicare con SSHDOS.

SSHDOS viene distribuito in due eseguibili differenti, a seconda della disponibilità o meno di una CPU i386. In generale, funziona sempre l'eseguibile '**`SSHDOS.EXE`**', anche se è molto lento:

`SSHDOS` [***opzioni***] ***utente nodo*** [***comando***]

In condizioni normali, non si usano le opzioni e nemmeno il comando da eseguire nell'elaboratore remoto; in questo modo viene chiesto di inserire la parola d'ordine, dopo la quale si ottiene di interagire con la shell. Eventualmente, per ottenere l'elenco delle opzioni disponibili, è sufficiente avviare l'eseguibile senza argomenti.

Se non si specificano opzioni particolari al riguardo, SSHDOS funziona emulando il comportamento di un terminale di tipo '**`xterm`**'; generalmente non è necessario cambiare questa impostazione con le opzioni.

Il pacchetto con cui è distribuito SSHDOS contiene anche un programma per l'uso del protocollo TELNET:

`TELNET` [***opzioni***] ***nodo*** [***porta***]

Dovrebbe essere disponibile anche un pacchetto separato per il protocollo SSH 2, ma potrebbe non essere completamente efficiente nella versione per CPU x86-16.

Bobcat (Lynx)

Bobcat è una raccolta di applicativi, organizzata attorno a una versione di Lynx [8] per Dos. Le licenze dei vari applicativi inseriti sono varie; tuttavia, il pacchetto più importante è proprio Lynx, che può funzionare anche da solo, per accedere ai servizi HTTP comuni.

Bobcat è ottenibile dall'indirizzo *http://www.fdisk.com/doslynx/bobcat.htm* . Eventualmente si può fare una ricerca con *http://www.google.com/search?q=bcat-e07.exe* .

Di tutti i file che compongono il pacchetto, sono sufficienti il programma '**`LYNX.EXE`**' e il file '`LYNX.CFG`'. Il secondo è il file di configurazione, in cui è bene definire la collocazione di alcuni file HTML (con estensione '`.HTM`'), che vengono usati quando si richiede la guida, altre informazioni e per accumulare eventualmente lo storico degli indirizzi richiesti. Inoltre, questa edizione di Lynx utilizza la libreria WATTCP, pertanto il programma si aspetta di trovare il file di configurazione '`WATTCP.CFG`' nella directory corrente.

PPRD: servente di rete per la stampa

software non libero: licenza Artistic

PPRD [9] è una piccola raccolta di programmi per la gestione di un servente di stampa secondo lo stile del demone '**`lpd`**'. Nelle situazioni in cui il sistema riesce a funzionare, permette di riutilizzare un vecchio PC, anche un XT, per questo scopo. Può essere ottenuto presso Simtel.Net all'indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/lan/pprd200.zip* .

PPRD si avvale della libreria WATTCP, pertanto può condividere la configurazione con altri programmi simili. In particolare, nel file di configurazione '`WATTCP.CFG`' si deve specificare la dimensione della memoria tampone (*buffer*) di trasmissione e ricezione.

```
TXBUFSIZE=8192
RXBUFSIZE=8192
```

Il programma eseguibile che svolge il lavoro è '**`PPRD.EXE`**', che viene avviato normalmente senza l'indicazione di alcuna opzione, purché il file '`WATTCP.CFG`' sia stato predisposto correttamente e collocato nella directory corrente:

PPRD [*opzioni*]

Salvo una diversa configurazione, il programma offre la stampante (o le stampanti) con un nome corrispondente a quello usato dal Dos per identificare il dispositivo: '**`lpt1`**', '**`lpt2`**', ...

Nell'elaboratore con un sistema GNU dal quale si vogliono inviare le stampe occorre sistemare il file '`/etc/printcap`' in modo adeguato. Quello che segue è un esempio in cui:

1. la stampante predefinita punta direttamente alla stampante remota, il cui nome è '**`lpt1`**';
2. la stampante '**`ps`**' utilizza un filtro che trasforma un documento PostScript in un file adatto alla stampante remota e poi lo ridirige alla stampante predefinita;
3. la stampante '**`tx`**' utilizza un filtro che trasforma un file di testo in stile Unix in un file di testo in stile Dos e poi lo ridirige alla stampante predefinita.

```
#
# /etc/printcap
#
lp:\
        :lp=:\
        :sd=/var/spool/lpd/lp:\
        :rm=192.168.1.15:\
        :rp=lpt1:\
        :mx#0:\
        :sf:\
        :sh:
#
ps:\
        :lp=/dev/null:\
        :sd=/var/spool/lpd/postscript:\
        :lf=/var/spool/lpd/postscript/log:\
        :if=/var/spool/lpd/postscript/input-filter:\
        :sh:\
        :sf:\
        :mx#0:
#
tx:\
        :lp=/dev/null:\
        :sd=/var/spool/lpd/text:\
        :lf=/var/spool/lpd/text/log:\
        :if=/var/spool/lpd/text/input-filter:\
        :sh:\
        :sf:\
        :mx#0:
```

Segue lo script usato come filtro di input per la stampa in PostScript. Lo script riceve i dati dallo standard input e attraverso Ghostscript lo trasforma in un file adatto per la stampa su una stampante a nove aghi tipo IBM-EPSON e dirige l'output verso la stampante predefinita, cioè quella remota.[10]

```
#!/bin/sh
/bin/grep -v '(%%' | /usr/bin/gs -q -dNOPAUSE -sPAPERSIZE=letter \
    -sDEVICE=eps9high -sOutputFile=- - | /usr/bin/lpr
```

Segue lo script usato come filtro di input per la stampa dei file di testo. Lo script riceve i dati dallo standard input e attraverso il programma '**`unix2dos`**' lo trasforma in un file di testo in cui ogni riga è terminata dalla sequenza *<CR><LF>* come richiesto dalle stampanti normali. L'output viene quindi diretto verso la stampante predefinita, cioè quella remota.

```
#!/bin/sh
/bin/cat | /usr/bin/unix2dos | /usr/bin/lpr
```

Se poi '**`unix2dos`**' non si comporta come previsto, si può realizzare un programma Perl:

```
#!/usr/bin/perl
##
## filtro-crlf.pl < <file-input> > <file-output>
##
$riga = "";
while ($riga = <STDIN>)
  {
    #
    # Elimina il codice di interruzione di riga finale.
    #
    chomp ($riga);
    #
    # Emette la riga con l'aggiunta di <CR> e <LF>.
    #
    print STDOUT ("$riga\r\n");
  };
```

Trout

Trout [11] è una versione Dos del noto Traceroute per sistemi Unix, che utilizza la libreria WATTCP; l'eseguibile che svolge il lavoro è '**`TROUT.EXE`**'.

Per ottenere il pacchetto, si può fare una ricerca per i file '`trtb01b.zip`' o '`trt-e01.exe`'.

Talk

Talk [12] è una versione Dos del noto programma con lo stesso nome per i sistemi Unix. Questa versione per Dos utilizza la libreria WATTCP e l'eseguibile che svolge il lavoro è '**`TALK.EXE`**'. Il pacchetto può essere recuperato presso l'indirizzo *http://http://users.libero.it/kht/mirror/half-mirror/dos/network/tcp.ip/wattcp/talk-13.zip* .

L'eseguibile '**`TALK.EXE`**' si comporta simultaneamente da servente e da cliente:

TALK [*opzioni*] [*utente@nodo* [*terminale*]]

Se non si indica l'utente e il nodo da contattare, si avvia il programma in attesa di chiamate, a cui poi viene data risposta, qualunque sia il nominativo utente che viene richiesto. Le opzioni disponibili sono poche; in particolare, '**`-l`**' consente di avere una registrazione della comunicazione in un file ('`TALK.LOG`'), mentre '**`-o`**' consente di richiedere l'uso di un protocollo più vecchio.

Durante il funzionamento, è possibile usare il tasto [*F1*] per ottenere una guida rapida all'uso dei comandi da tastiera; in particolare, la

combinazione [*Alt s*], consente di cambiare la modalità visiva della comunicazione (a schermo unico o a schermo diviso).

La configurazione con il file ‘`WATTCP.CFG`’ prevede l’aggiunta di direttive specifiche, che comunque non sono indispensabili.

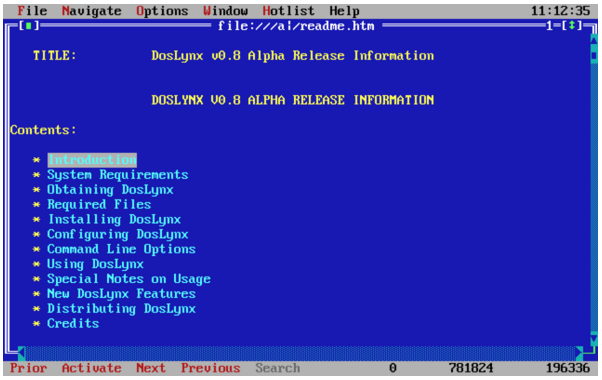
DosLynx

« software didattico, non libero

DosLynx [13] è un programma di navigazione a caratteri, ma relativamente completo, da utilizzare insieme a un driver di pacchetto per il TCP/IP (non fa uso della libreria WATTCP, pertanto ha una configurazione indipendente). Può essere ottenuto presso *http://www.fdisk.com/doslynx/doslynx.htm* . Per quanto possibile, questo applicativo è molto accurato, per esempio permette l’uso del mouse.

L’eseguibile è precisamente ‘**`DOSLYNX.EXE`**’ che si avvia senza l’indicazione di argomenti particolari; ma prima di poter essere utilizzato occorre predisporre il file ‘`DOSLYNX.CFG`’.

Figura u59.21. La guida interna di DosLynx.



DosLynx permette anche l’invio di messaggi di posta elettronica, ma non la loro ricezione o lettura.

La configurazione di DosLynx avviene attraverso il file ‘`DOSLYNX.CFG`’, collocato nella directory corrente nel momento in cui si avvia il programma. In questo file, per prima cosa deve essere definito l’indirizzo IP e la maschera di rete (*netmask*).

```
my_ip=192.168.1.15
netmask=255.255.255.0
```

Quindi occorre indicare l’indirizzo del router (gateway) e del servente DNS anche se in realtà possono non esistere nella rete locale che si utilizza.

```
gateway=192.168.1.1
nameserver=192.168.1.1
```

Viene specificato quindi il dominio e il nome dell’elaboratore locale.

```
domainslist="brot.dg"
hostname=dos
```

Per il resto, questo file di configurazione viene già fornito con un esempio molto ben commentato. Vale comunque la pena di indicare:

- l’attivazione del collegamento con l’esterno;
- il proprio indirizzo di posta elettronica, che viene utilizzato come mittente per i messaggi inviati;
- l’indicazione dell’elaboratore a cui fare riferimento per l’inoltro dei messaggi di posta elettronica inviati, attraverso il protocollo SMTP;
- l’indicazione dell’elaboratore a cui fare riferimento per l’accesso a NNTP (news).

```
networked=YES
mailaddr=daniele@dinkel.brot.dg
smtphost=192.168.1.1
nntphost=192.168.1.1
```


NCSA Telnet

« NCSA Telnet [14] è una piccola raccolta di programmi da utilizzare insieme a un driver di pacchetto per il TCP/IP (si tratta di programmi autonomi dalla libreria WATTCP). Può essere ottenuta presso Simtel.Net all’indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/ncsatlnt/* .

Nelle sezioni seguenti vengono descritti solo alcuni dei programmi di questa raccolta. Prima di poterli utilizzare occorre predisporre il file ‘`CONFIG.TEL`’.

L’ultima versione di questa raccolta dovrebbe essere la 2.3.08 che però sembra avere qualche problema, in particolare non può essere utilizzata quando si abilita il *Path MTU discovery* durante la compilazione del kernel Linux. La versione 2.3.07.4 (precedente) dovrebbe essere esente da questo difetto. Inoltre, alcune versioni precedenti alla 2.3.08, compresa la 2.3.07.4, contengono più programmi di servizio accessori, come un programma per il ping e uno per il tracciamento dell’instradamento. Per trovare la versione 2.3.07.4 si può provare con: *http://www.google.com/search?q=tel23074.zip* .

File «CONFIG.TEL»

« ‘`CONFIG.TEL`’ è un file di testo contenente l’indicazione della configurazione del gruppetto di programmi che compongono il pacchetto Telnet NCSA. Per prima cosa deve essere definito l’indirizzo IP e la maschera di rete (*netmask*).

```
myip=192.168.1.15
netmask=255.255.255.0
```

Quindi occorre definire le caratteristiche del driver di pacchetto utilizzato, che a loro volta dipendono dal tipo di scheda di rete. L’esempio seguente riguarda il caso del driver di pacchetto Crynwr per la scheda NE2000 (‘**`NE2000.COM 0x7e 0x0b 0x300`**’). Occorre fare attenzione alla voce ‘**`ioaddr=`**’ che non si riferisce a un indirizzo di I/O, ma all’IRQ software del driver di pacchetto.

```
hardware=packet
interrupt=11
ioaddr=0x7e
```

Segue una serie di altre informazioni, in particolare sono interessanti le seguenti.

```
myname=dos.brot.dg
termtype="vt100"
keyfile=".\keymap.tel"
services=".\services.tel"
ftp=yes
ftpwrt=yes
passfile=".\password.tel"
```

Quindi vengono richieste le informazioni sui nodi che possono essere contattati. Vengono inizialmente indicate delle caratteristiche generali predefinite, quindi i dati particolari di nodi determinati.

```
name=default
# Segue una serie di caratteristiche predefinite
# ...
# Inizia la definizione dell'elaboratore «dinkel.brot.dg»
name=dinkel.brot.dg
hostip=192.168.1.1
gateway=0       # Non è un router
nameserver=1    # È un Name Server
# Inizia la definizione dell'alias «dinkel»
name=dinkel
copyfrom=dinkel.brot.dg
```

File «KEYFILE.TEL»

« All’interno del file ‘`CONFIG.TEL`’, con la voce ‘**`keyfile=`**’, viene dichiarato il nome e la collocazione di un file di configurazione della tastiera. Lo scopo di questo file è definire una corrispondenza tra tasti premuti e segnali inviati. Telnet NCSA fornisce già questo file e ha il nome ‘`KEYFILE.TEL`’.

Per poter conoscere i codici a cui corrispondono i tasti della propria tastiera, si può utilizzare il programma ‘**`SCANCHEK.EXE`**’.

In generale, non conviene modificare il file originale, piuttosto, è meglio tentare diversi tipi di configurazione di terminale assegnando

un valore opportuno alla variabile di ambiente '**TERM**' di GNU/Linux o alla voce '**termtype=**' del file 'CONFIG.TEL'. si possono provare, in particolare, i valori '**vt100**' e '**vt220**'.

File «PASSWORD.TEL»

Il programma '**TELBIN.EXE**' può funzionare anche come un semplice servente FTP per accessi singoli.[15]

Per questo, è necessario definire un file contenente informazioni sugli utenti e sui loro permessi di accesso. Il nome e la posizione di questo file viene definito all'interno di 'CONFIG.TEL', con la voce '**passfile=**' e di solito si tratta di 'PASSWORD.TEL'.

Per crearlo o modificarlo, conviene utilizzare il programma '**TELPASS.EXE**', per esempio nel modo seguente. Il programma stesso suggerisce le operazioni da compiere.

```
C:\NCSATELN> telpass password.tel [Invio]
```

File «SERVICES.TEL»

'SERVICES.TEL' è il file dei servizi di rete ed è analogo al file '/etc/services' (32.8.2). Viene già fornito configurato correttamente.

File «TELBIN.EXE»

```
TELBIN [nodo]
```

Il programma '**TELBIN.EXE**' è il più importante di questo gruppo, essendo quello che permette di attivare una connessione TELNET con un elaboratore GNU/Linux o un altro Unix. Se il programma non riesce a connettersi con l'elaboratore indicato come argomento, o se questo non viene indicato, si avvia come servente FTP e accetta una sola connessione alla volta.

Quando si vuole utilizzare '**TELBIN.EXE**' come servente FTP occorre predisporre il file degli utenti, che solitamente è 'PASSWORD.TEL'.

File «FTPBIN.EXE»

```
FTPBIN [nodo]
```

'**FTPBIN.EXE**' è il secondo programma come importanza. Si tratta di un semplice cliente FTP abbastanza funzionante. I comandi che mette a disposizione sono i soliti per questo tipo di programma; per ottenere aiuto si può utilizzare il punto interrogativo ('**?**').

File «FINGER.EXE»

```
FINGER [utente]@nodo
```

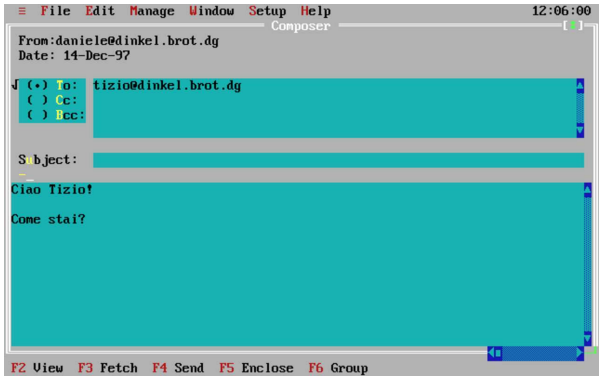
Il programma '**FINGER.EXE**' permette di ottenere informazioni sugli utenti connessi in un elaboratore determinato. Il risultato di questa interrogazione è analogo a quello del suo omonimo negli ambienti Unix.

POPMail

POPMail [16] è un ottimo programma per la gestione della posta elettronica attraverso la connessione con un servizio POP2 o POP3. Può essere ottenuto presso Simtel.Net all'indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/pktdrvr/popml322.zip* .

La configurazione viene fatta attraverso il programma stesso e non richiede la preparazione di alcun file.

Figura u59.30. La composizione di un messaggio di posta elettronica attraverso POPMail.



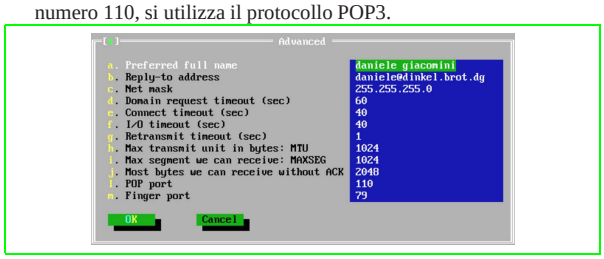
POPMail si compone di un solo eseguibile monolitico: '**POPMAIL.EXE**'. Tutte le sue funzionalità sono incorporate in questo, compresa la configurazione. Appena si avvia il programma si ottiene un'interfaccia amichevole che permette l'uso del mouse.

Menù «Setup»

La configurazione del programma si definisce attraverso le funzioni del menù *Setup*. In particolare è importante la voce *Network*, attraverso cui si accede a una maschera per la definizione degli indirizzi e dei nomi utilizzati. In questa fase, è importante stabilire il tipo di protocollo che si intende utilizzare. Questo lo si fa attraverso l'indicazione della porta di comunicazione. Quella predefinita è 109 corrispondente a POP2, altrimenti si può utilizzare la porta 110 in modo da collegarsi a un servizio POP3.

Figura u59.31. La finestra principale della configurazione di POPMail. Si può osservare che il nodo da specificare alla voce *Host Computer* è quello che fornisce il servizio SMTP, mentre subito sotto è richiesto l'indirizzo dell'elaboratore locale.

Figura u59.32. Selezionando il pulsante ADVANCED dalla finestra principale di configurazione, si ottiene questa finestra di informazioni aggiuntive. La selezione del tipo di protocollo dipende dal numero di porta selezionato. In questo caso, essendo il numero 110, si utilizza il protocollo POP3.



Menù «Window»

Una volta definita la configurazione, si può iniziare a utilizzare il programma per ricevere e spedire posta. Esistono tre finestre: una per la composizione dei messaggi, un'altra per la loro lettura e l'ultima per le operazioni di «taglia-copia-incolla». Per passare da una finestra all'altra, occorre richiamare questo menù.

Menù «=»

Il menù dell'applicazione, quello precedente a *File*, permette di accedere a funzionalità aggiuntive e molto utili. Si può utilizzare una sessione TELNET in una finestra, si può ottenere la risoluzione di indirizzi IP e si può eseguire il ping.

Figura u59.33. Una sessione TELNET attraverso POPMail.

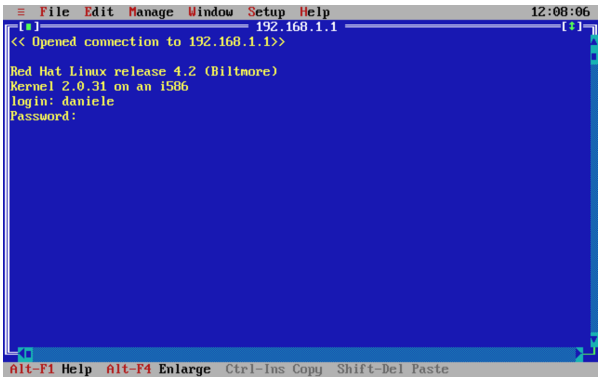


Figura u59.34. La presenza di una funzione di ping completa l'applicativo POPMail.

PCroute

PCroute permette di trasformare un vecchio PC (i286 o inferiore) in un router IPv4. Può essere ottenuto presso Simtel.Net all'indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/network/pcrte224.zip* .

Nell'archivio che viene distribuito, è presente il sorgente e diverse versioni compilate, per l'uso delle schede di rete più comuni nel passato. Tra queste versioni già pronte ne esiste una in grado di utilizzare i driver di pacchetto descritti all'inizio di questo capitolo. Gli esempi che vengono mostrati qui si riferiscono all'utilizzo dei driver di pacchetto.

Configurazione dei driver di pacchetto

Se si decide di utilizzare la versione già compilata per i driver di pacchetto, cioè 'PKTPKT.EXE', è necessario prima configurare i driver di pacchetto, poi si può pensare alla configurazione di PCroute.

La versione precompilata, 'PKTPKT.EXE', prevede l'utilizzo di due indirizzi di interruzione (*interrupt*) software per comunicare con i driver di pacchetto, 60_{16} e 61_{16}, dove il primo si riferisce alla prima interfaccia e l'altro alla seconda.

Supponendo di disporre di schede di rete compatibili NE2000, che utilizzino rispettivamente le risorse IRQ 10 e I/O 280_{16}, IRQ 11 e I/O 300_{16}, la configurazione dei driver di pacchetto dovrebbe essere la seguente:

```
ne2000 0x60 0x0a 0x280
ne2000 0x61 0x0b 0x300
```

Configurazione di PCroute

Per fare funzionare PCroute è necessario l'eseguibile 'PCROUTE.EXE'; nel caso di utilizzo dei driver di pacchetto, si tratta di 'PKTPKT.EXE'. Inoltre serve anche 'CONFIG.EXE', per generare il file di configurazione di PCroute.

Si suppone che la prima scheda sia inserita nella rete 192.168.1.0 e che abbia l'indirizzo 192.168.1.254; inoltre si suppone che la seconda sia nella rete 192.168.2.0 con l'indirizzo 192.168.2.254. Non si

prevede la necessità di indicare altri instradamenti per mezzo di altri router.

```
C:\PCROUTE> CONFIG [Invio]

  This program creates/edits the pcroute.cfg file
```

Inizia una configurazione interattiva, a cominciare dalle indicazioni riferite alla prima interfaccia, cioè quella collegata al driver di pacchetto attraverso l'indirizzo IRQ 60_{16}.

```
  Configuring an interface

Address for the interface [0.0.0.0] ? 192.168.1.254 [Invio]

Subnet mask for the interface [255.255.255.0] ? 255.255.255.0
[Invio]


  Flag Meanings (if set)
      Bit 0 (1h)  - Don't send routing updates out this interface
      Bit 1 (2h)  - Don't listen to routing updates from this interface
      Bit 2 (4h)  - Proxy Arp for all subnets
      Bit 3 (8h)  - Turn off directed broadcasts
      Bit 4 (10h) - Turn off the issuing of ICMP redirects
      Bit 5 (20h) - Broadcast using old (0's) format

Flags (HEX) for the interface [0H] ? 0H [Invio]

Routing Metric (HEX) for the interface [1H] ? 1H [Invio]
```

A questo punto si passa alla configurazione della seconda interfaccia, cioè quella collegata al driver di pacchetto attraverso l'indirizzo IRQ 61_{16}.

```
  Configuring an interface

Address for the interface [0.0.0.0] ? 192.168.2.254 [Invio]

Subnet mask for the interface [255.255.255.0] ? 255.255.255.0
[Invio]


  Flag Meanings (if set)
      Bit 0 (1h)  - Don't send routing updates out this interface
      Bit 1 (2h)  - Don't listen to routing updates from this interface
      Bit 2 (4h)  - Proxy Arp for all subnets
      Bit 3 (8h)  - Turn off directed broadcasts
      Bit 4 (10h) - Turn off the issuing of ICMP redirects
      Bit 5 (20h) - Broadcast using old (0's) format

Flags (HEX) for the interface [0H] ? 0H [Invio]

Routing Metric (HEX) for the interface [1H] ? 1H [Invio]
```

Gli instradamenti sulle reti cui sono connesse le interfacce vengono definiti in modo automatico. Si decide di non indicare altri instradamenti particolari.

```
  If you wish to configure static routes do so here.  To stop type a '.'

    Flag Meanings (if set)
        Bit 0 (1h)  - Local route, do not propagate it
        Bit 1 (2h)  - Transient route, subject to RIP protocol

Network [0.0.0.0] ? . [Invio]
```

Da questo punto non si seleziona alcuna opzione particolare.

```
  If you wish to forward bootp packets please enter the address
    of the address to forward it to.  This address can be a
    directed broadcast.  0.0.0.0  means don't forward

Address to forward bootp packets [0.0.0.0] ? 0.0.0.0 [Invio]


  Once PCroute boots up, it sends all log messages to a network
    host running a BSD UNIX syslogd daemon.  To disable
    logging enter 0.0.0.0

Host to send loging info to [0.0.0.0] ? 0.0.0.0 [Invio]


  Mask Meanings (0 = Log, 1 = Don't log)
      Bit 0 (1h)  - System
      Bit 1 (2h)  - Routing
      Bit 2 (4h)  - Monitor
      Bit 3 (8h)  - Localtalk

Logging mask for this router [0H] ? 0H [Invio]


  There are 8 routing 'levels' supported
      0 - Emergency     1 - Alert     2 - Critical     3 - Error
      4 - Warning       5 - Notice    6 - info         7 - Debug
  Only messages with a level less than the logging level are sent

Logging level [0H] ? 0H [Invio]
```

A questo punto la configurazione termina e ne viene generato il file ‘PCROUTE.CFG’.

Conclusione

PCroute, per funzionare richiede solo l’avvio dell’eseguibile (‘**PCROUTE.EXE**’), che ha la necessità di trovare il file ‘PCROUTE.CFG’ nella directory corrente. Dopo l’avvio, l’elaboratore risulta bloccato, essendo destinato esclusivamente alla funzione di instradamento.

La documentazione di PCroute spiega meglio come gestire le varie opzioni, che nell’esempio sono state evitate semplicemente, descrivendo anche come sfruttare la possibilità di tenere sotto controllo il funzionamento di PCroute attraverso il registro di sistema di un elaboratore come GNU/Linux.

Riferimenti

- Erick Engelke, *WATTCP*
  http://www.wattcp.com/
- Smash-Co Communications, *TCP/IP for MS-DOS*
  http://www.smashco.com/wattcp.asp
- *WATTCP*
  http://www.wattcp.com/
  http://www.wattcp.com/wat1104.zip
- *The U-M Software Archive*
  http://www.umich.edu/~archive/msdos/communications/wattcp/
  http://www.umich.edu/~archive/msdos/communications/packet/
- *Arachne labs*
  http://www.arachne.cz/
- *Arachne GPL*
  http://home.hetnet.nl/~ba8tian/arachne/arachne.htm
  http://home.hetnet.nl/~ba8tian/arachne/175-gpl/ar175.htm
  http://home.hetnet.nl/~ba8tian/arachne/175-gpl/a175gp75lf.zip
- *Using the Internet (or an IP Network) from a MSDOS Machine*
  http://www.geocities.com/SiliconValley/Park/2884/dosint.htm

[1] **Crynwr packet driver collection** GNU GPL

[2] **WATTCP** software non libero: non è consentita la distribuzione di versioni modificate e non è consentita la commercializzazione

[3] Per indicare più serventi DNS, è sufficiente usare la direttiva ‘**NAMESERVER**’ ripetutamente.

[4] **WATTCP apps** software non libero: non può essere commercializzato

[5] **ABC-nslookup** UCB BSD

[6] **MiniTelnet** software libero con licenza speciale

[7] **SSHDOS** GNU GPL

[8] **Lynx** GNU GPL

[9] **PPRD** software non libero: licenza Artistic

[10] Si suppone di usare carta a modulo continuo, pertanto viene indicato il formato lettera (11 in).

[11] **Trout** software gratuito senza sorgenti

[12] **Talk** GNU GPL

[13] **DosLynx** software didattico, non libero

[14] **NCSA Telnet** dominio pubblico

[15] Non è il caso di fare affidamento su questa funzionalità di ‘**TELBIN.EXE**’ perché non è perfettamente funzionante.

[16] **POPMail** software gratuito non modificabile e senza sorgenti

Dos PPP

Composizione 371
Configurazione e script 371
Opzioni particolari per il PPP 372
Connessione in pratica 372

Per realizzare una connessione PPP con un sistema Dos, è necessario un driver di pacchetto speciale, più o meno derivato dal demone ‘**pppd**’ tradizionale dei sistemi Unix. Dal momento che di solito si usa una connessione PPP attraverso un modem e una linea commutata, è necessario anche un programma analogo a ‘**chat**’ per attivare il modem e per superare la procedura iniziale.

Esistono diversi programmi per Dos in grado di gestire in qualche modo il protocollo PPP, ma sembra essere solo la realizzazione DOS PPP [1] ad avere il pregio di essere semplice e compatibile con i driver di pacchetto Ethernet.

L’archivio contenente DOS PPP dovrebbe essere accessibile dall’indirizzo *ftp://ftp.simtel.net/pub/simtelnet/msdos/pktdrvr/dosppp05.zip* .

Composizione

Il pacchetto di distribuzione di DOS PPP si compone di alcuni programmi, dove i più importanti sono:

- ‘**EPPPD.EXE**’
  il programma residente in memoria che svolge il ruolo del demone PPP tradizionale, emulando una scheda Ethernet;
- ‘**CHAT.EXE**’
  il programma utilizzato attraverso ‘**EPPPD.EXE**’ per comandare il modem.

Questi due programmi emulano il più possibile i loro progenitori per Unix: ‘**pppd**’ e ‘**chat**’, tenendo conto di alcuni aggiustamenti dovuti alle carenze del Dos.

Configurazione e script

La configurazione di DOS PPP segue idealmente quella del demone Unix, con la differenza che i file hanno nomi e collocazioni differenti. Considerando che si tratta di sistemi Dos, si possono anche semplificare un po’ le cose, come descritto nel seguito.

- Il file ‘PPPD.CFG’, collocato nella stessa directory in cui si trova l’eseguibile ‘**EPPPD.EXE**’, oppure nella directory corrente, rappresenta in pratica quello che tradizionalmente è il file ‘/etc/ppp/options’.
  Naturalmente, si possono usare anche opzioni della riga di comando, le quali prevalgono sulle opzioni fissate con il file di configurazione.
- I file ‘PPPDCOM*n*.CFG’, collocati nella directory corrente, permettono di indicare opzioni specifiche per ogni porta seriale: ‘PPPDCOM1.CFG’ per ‘COM1:’, ‘PPPDCOM2.CFG’ per ‘COM2:’, ecc. In questo modo si emulano i file di configurazione ‘/etc/ppp/options/options.tty*’ tradizionali nei sistemi Unix.

Dai file di configurazione è esclusa la presenza di qualcosa che serva per contenere i segreti PAP e CHAP. Per queste informazioni sono state aggiunte delle opzioni da inserire nei file di configurazione normali.

Dal momento che non c’è un modo migliore per fare sapere quali sono le caratteristiche IP della connessione che si instaura, viene generato automaticamente lo script ‘**IP-UP.BAT**’, il cui unico scopo è quello di inizializzare alcune variabili di ambiente:

```
SET MYIP=indirizzo_ip_locale
SET REMIP=indirizzo_ip_della_controparte
SET NETMASK=maschera_di_rete
SET PEERMRU=valore_MRU_della_controparte
```

Questo dovrebbe facilitare la realizzazione di un altro script che generi al volo i file di configurazione degli applicativi che si intendono usare. Per esempio, volendo realizzare il file di configurazione '`WATTCP.CFG`' per i programmi che incorporano la libreria WATTCP, si potrebbe procedere come si vede nell'esempio seguente:

```
CALL IP-UP.bat
ECHO MY_IP=%MYIP% > WATTCP.CFG
ECHO GATEWAY=%REMIP% >> WATTCP.CFG
ECHO NETMASK=%NETMASK% >> WATTCP.CFG
ECHO NAMESERVER=195.210.91.1 >> WATTCP.CFG
ECHO MSS=512 >> WATTCP.CFG
```

Si può osservare che lo script, oltre a tradurre le variabili di ambiente in direttive del file '`WATTCP.CFG`', aggiunge anche le direttive necessarie per definire il servente e per definire la dimensione massima dei segmenti di pacchetto.

Su '**`CHAT.EXE`**' non c'è nulla di speciale, tranne il fatto che questo programma non può funzionare da solo, ma deve trovarsi sotto il controllo di '**`EPPPD.EXE`**'. Le opzioni sono molto simili alla versione originale per i sistemi Unix. In generale vale la pena di utilizzare l'opzione '**`-v`**' per vedere cosa succede durante l'avvio della connessione.

Opzioni particolari per il PPP

Il programma '**`EPPPD.EXE`**' accetta la maggior parte delle opzioni delle vecchie edizioni di '**`pppd`**' per i sistemi Unix. Per verificare quali sono le opzioni disponibili basta leggere la documentazione allegata, che riproduce la pagina di manuale relativa.

- `user` ***utente***

  `passwd` ***parola_d'ordine***

  DOS PPP è in grado di gestire esclusivamente l'autenticazione PAP, ma senza l'ausilio di un file dei segreti. In pratica, si fa uso delle opzioni '**`user`**' e '**`passwd`**', con le quali si fornisce il nominativo utente e la parola d'ordine, senza altre specifiche.

- `pktvec` ***irq***

  Dal momento che si tratta di driver di pacchetto, DOS PPP si avvale di un IRQ software che può essere scelto esplicitamente, oppure può essere definito automaticamente dal programma. L'opzione '**`pktvec`**' permette di fissare il valore di tale IRQ, assegnando valori esadecimali nella forma '**`0x`*`nn`***'. Il valore predefinito usuale è 60_{16}.

Connessione in pratica

Si suppone di avere la possibilità di collegarsi a un servizio di accesso a Internet che ha le caratteristiche seguenti:

- telefono 0987 6543210;
- utenza '**`tizio`**';
- parola d'ordine '**`asdfghjk`**';
- indirizzo IP del DNS primario 123.123.123.1.

Inoltre, si utilizza la prima porta seriale, ovvero '`COM1:`', che viene configurata per una velocità di 57600 bit/s. Si realizza il file '`PPPD.CFG`' con il contenuto seguente:

```
com1
57600
user tizio
passwd asdfghjk
connect "chat -v '' ATZ OK ATX3 OK ATDT9876543210 CONNECT '' "
```


In questo modo, quando si avvia '**`EPPPD.EXE`**', questo avvia prima '**`CHAT.EXE`**' in modo da inizializzare il modem, comporre il numero telefonico e attendere la connessione; successivamente, l'autenticazione avviene attraverso il protocollo PAP.

Si può osservare l'opzione '**`-v`**' di '**`CHAT.EXE`**', che serve per vedere i messaggi scambiati tra questo programma e il modem, durante le operazioni. La conoscenza di questi dettagli serve per correggere eventualmente la stringa, in base al comportamento effettivo del modem.[2]

Una volta instaurata la connessione, '**`EPPPD.EXE`**' crea il file '`IP-UP.BAT`', che può essere sfruttato come è già stato visto in precedenza da un altro script che generi i file di configurazione necessari agli altri applicativi, specificando così anche il DNS primario e la dimensione massima del segmento (MSS). L'esempio seguente mostra uno script necessario a generare un file di configurazione per gli applicativi che usano la libreria WATTCP:

```
CALL IP-UP.bat
ECHO MY_IP=%MYIP% > WATTCP.CFG
ECHO GATEWAY=%REMIP% >> WATTCP.CFG
ECHO NETMASK=%NETMASK% >> WATTCP.CFG
ECHO NAMESERVER=123.123.123.1 >> WATTCP.CFG
ECHO MSS=512 >> WATTCP.CFG
```

Per concludere la connessione, si usa il programma '**`TERMIN.COM`**', che viene distribuito anche assieme a DOS PPP, ma per questo occorre conoscere l'indirizzo IRQ software utilizzato da '**`EPPPD.EXE`**'. Per esempio, se si tratta dell'indirizzo IRQ 60_{16} (quello predefinito), basta procedere come segue:

```
C:>TERMIN 0x60 [Invio]
```

[1] **DOS PPP** software non libero: non è consentita la modifica e nemmeno la commercializzazione

[2] Naturalmente, nello stesso modo si potrebbe realizzare un accesso di tipo tradizionale, in cui sia '**`CHAT.EXE`**' a inviare il nominativo utente e la parola d'ordine. Tuttavia, è sempre meno probabile che un fornitore di accesso a Internet utilizzi ancora tale vecchia procedura.

Preparazione 376
Configurazione con il file «AUTOEXEC.NET» 376
Driver di pacchetto 377
Avvio del sistema NOS (NET.EXE) 377
Conclusione del funzionamento del sistema NOS 378
Guida interna 378
Interfacce, instradamento e nomi 378
Comandi «attach» e «detach» 378
Comando «ifconfig» 379
Comando «route» 380
Comando «domain» 380
Gestione delle sessioni 381
Attività nel sistema locale 381
Gestione della rete e delle connessioni 382
Indirizzi IP e indirizzi MAC 382
Ping e instradamento 382
Varie 383
NOS come cliente 383
Cliente TELNET 383
Cliente FTP 383
NOS come servente 384
Registrazione degli eventi 384
Servente FTP 384
NOS come router IPv4 385

NOS [1] è una sorta di sistema operativo per le reti IPv4 nato per soddisfare le esigenze dei radioamatori. Se si considerano l'età e il fatto che funziona perfettamente su un sistema operativo Dos, si tratta di un applicativo eccezionale quando si dispone di hardware molto vecchio. La sigla KA9Q è il nominativo da radioamatore dell'autore originale di questo programma, Phil Karn, ma spesso si fa riferimento a questo software indifferentemente con le sigle KA9Q, NOS o qualcosa che termina per ∗NOS.[2]

Esistono diverse interpretazioni del sistema NOS-KA9Q; probabilmente il riferimento migliore per ottenere il materiale necessario è il deposito Simtel.Net, che ospita una directory apposita per questo: *ftp://ftp.simtel.net/pub/simtelnet/msdos/tcpip/* . In particolare è necessario prelevare il file contenente l'eseguibile '**`NET.EXE`**', che potrebbe avere un nome simile a '`e920603.zip`' (dove il numero corrisponde alla data ed eventualmente potrebbe essere sostituito da una versione più recente) e poi conviene prelevare altri file per ottenere della documentazione: '`intronos.zip`', '`ka9qbgn.zip`' e '`nos_slfp.zip`' (questo ultimo file può essere utile soprattutto per vedere come potrebbe essere effettuata una connessione PPP attraverso la porta seriale e il modem).

La versione 920603, corrispondente al file '`e920603.zip`', è adatta ad architetture x86-16. Probabilmente ciò vale anche per qualche versione più recente, ma si deve fare attenzione: la versione 951123 è fatta per x86-32. Se non si riesce a trovare una versione del programma '**`NET.EXE`**' abbastanza vecchia, si può provare a usare quella contenuta nel pacchetto '`nos_slfp.zip`'. In questo capitolo si fa riferimento a una versione di NOS per architetture modeste (i286 o inferiori).

NOS, una volta avviato, prende il controllo del sistema e i comandi che si impartiscono sono interpretati da questo, senza passare per il Dos sottostante. Anche per questa ragione si introduce l'uso di NOS in un capitolo separato, rispetto a quello già dedicato agli applicativi Dos (capitolo u59).

Qui si mostrano le caratteristiche «normali» di NOS, nel senso che di questo sistema di rete sono state realizzate un'infinità di varianti. Evidentemente, il NOS che si può trovare può corrispondere o meno alle caratteristiche che vengono descritte qui. Se il pacchetto NOS che si trova contiene qualche file di documentazione, conviene leggerlo per verificare che tutto corrisponda a quanto previsto.

Preparazione

Anche se non si intendono sfruttare a fondo tutte le possibilità di NOS, conviene creare tutte le directory previste da questo mini sistema di rete. Se non si vuole fare fatica nella configurazione, conviene predisporre quelle seguenti, che riguardano le versioni «normali» di NOS:

```
C:\SPOOL
C:\SPOOL\HELP
C:\SPOOL\MAIL
C:\SPOOL\MQUEUE
C:\SPOOL\RQUEUE
C:\SPOOL\NEWS
```

Come si può intuire, si tratta di spazi predisposti per la gestione della posta elettronica; cosa che comunque non viene mostrata in questo capitolo.

Volendo utilizzare una posizione diversa, nello stesso disco o in un altro, occorre almeno mantenere la stessa struttura; per esempio come nel modo seguente, tenendo conto che occorre avviare il programma '**`NET.EXE`**' specificando questa variante nelle opzioni.

```
D:\NOS\SPOOL
D:\NOS\SPOOL\HELP
D:\NOS\SPOOL\MAIL
D:\NOS\SPOOL\MQUEUE
D:\NOS\SPOOL\RQUEUE
D:\NOS\SPOOL\NEWS
```

Configurazione con il file «AUTOEXEC.NET»

Nella directory utilizzata come punto di inizio della gerarchia del sistema NOS, va collocato il file di configurazione '`AUTOEXEC.NET`'. Questo rappresenta semplicemente una sequenza di comandi NOS da eseguire prima di mostrare l'invito all'utente. È abbastanza importante predisporre questo file, per non dover ogni volta ridefinire la configurazione delle interfacce e gli instradamenti relativi.

Ovviamente, per sapere come predisporre questo file occorre conoscere i comandi del sistema NOS. Per cominciare si tenga presente che sono ammessi i commenti prefissati dal simbolo '`#`' e terminati dalla fine della riga in cui appaiono; inoltre, se si utilizza un elaboratore appartenente alla famiglia «AT», cioè quelli che hanno un'architettura i286 o superiore, può essere utile indicare il comando '**`isat on`**'. Per il momento si osservi l'esempio seguente, che si riferisce all'uso di una scheda di rete gestita attraverso un driver di pacchetto di quelli descritti nel capitolo precedente (u59).

```
# Se non si tratta di un elaboratore compatibile IBM AT (o superiore),
# la riga seguente deve essere commentata o eliminata.
#isat on

# Configurazione dell'interfaccia di rete utilizzando il
# riferimento al packet driver (il nome ethernet0 viene stabilito qui,
# e non si tratta di una convenzione di NOS).
attach packet 0x7e ethernet0 8 1500

# Definizione dell'indirizzo IP dell'interfaccia di rete.
ifconfig ethernet0 ipaddress 192.168.1.10
ifconfig ethernet0 netmask 255.255.255.0

# Instradamento.
route add 192.168.1.0/24 ethernet0
route add default ethernet0 192.168.1.254

# DNS
domain addserver 192.168.1.1
domain suffix brot.dg.

# Servizi abilitati.
start discard
start echo
start finger
start ftp
```


Driver di pacchetto

Il sistema NOS richiede per funzionare che le interfacce di rete da utilizzare siano controllate da un driver di pacchetto, tranne nei casi in cui è in grado di gestirsele da solo. NOS può utilizzare i driver di pacchetto già mostrati nel capitolo u59 e altri specifici, come nel caso del file '`nos_slfp.zip`' che contiene il necessario per gestire una connessione PPP partendo dal controllo della porta seriale e del modem.

Per non appesantire troppo la presentazione del sistema NOS, vengono mostrati solo esempi che fanno riferimento a una scheda di rete gestita attraverso un driver di pacchetto configurato in modo da utilizzare l'indirizzo IRQ $7E_{16}$ per comunicare con le applicazioni. Volendo fare il solito esempio della scheda NE2000 configurata per usare l'indirizzo IRQ 11 e la porta di I/O 300_{16}, si tratta di usare il comando seguente:

```
NE2000.COM 0x7e 0x0b 0x300
```

Avvio del sistema NOS (NET.EXE)

Tutto il sistema NOS è inserito in un solo eseguibile Dos: '**`NET.EXE`**'. All'avvio del programma può essere conveniente utilizzare qualche opzione.

`NET` [`-b`] [`-s` ***n_porte***] [`-d` ***directory_nos***] [***file_configurazione***]

Dopo l'avvio, il sistema NOS mostra alcune informazioni riferite alla versione e quindi l'invito a inserire dei comandi:

```
KA9Q NOS version 910618 -> 911007 (ghm/was)
Copyright 1990 by Phil Karn, KA9Q
net> _
```

Qui viene mostrata una versione particolarmente vecchia del programma; se si trattasse di un'edizione specifica per microprocessori i386 o superiori, tale informazione apparirebbe tra quelle che precedono l'invito.

Tutti i comandi che vengono descritti nelle sezioni successive devono essere impartiti al sistema NOS attraverso l'invito '**`net>`**', oppure possono essere collocati nel file di configurazione (di solito '`AUTOEXEC.NET`').

Opzione o argomento	Descrizione
`-b`	Con questa opzione si costringe NOS ad aggiornare lo schermo attraverso le funzioni del BIOS, anziché accedendo direttamente alla memoria video. Ciò rallenta le operazioni, ma può essere necessario in alcune circostanze, quando si vede che la visualizzazione sullo schermo non funziona come ci si aspetterebbe.
`-s` ***n_porte***	In condizioni normali, NOS gestisce un massimo di 40 connessioni (un massimo di 40 porte). Se si vuole modificare questo valore si può intervenire con l'opzione '**`-s`**'.
`-d` ***directory_nos***	Se la struttura di directory che richiede NOS si trova a partire da una posizione differente dalla directory radice del disco '`C:`', l'opzione '**`-d`**' permette di indicarlo esplicitamente.
file_configurazione	Se si vuole indicare esplicitamente il file di configurazione (che di solito dovrebbe essere '***directory_nos***`\AUTOEXEC.NET`'), questo può essere inserito come ultimo argomento della riga di comando, senza l'indicazione di un'opzione apposita.

Segue la descrizione di alcuni esempi.

- `C:\> NET` [*Invio*]

  Avvia il sistema NOS utilizzando la gerarchia che si articola a partire dalla directory radice del disco '`C:`' ('`C:\SPOOL\*`') e il file di configurazione '`C:\AUTOEXEC.NET`'.

- `C:\> NET -d C:\NOS` [*Invio*]

  Avvia il sistema NOS utilizzando la gerarchia che si articola a partire dalla directory '`C:\NET\`' ('`C:\NET\SPOOL\*`') e il file di configurazione '`C:\NET\AUTOEXEC.NET`'.

- `C:\> NET -d C:\NOS C:\NOS.RC` [*Invio*]

  Avvia il sistema NOS utilizzando la gerarchia che si articola a partire dalla directory '`C:\NET\`' ('`C:\NET\SPOOL\*`') e il file di configurazione '`C:\NOS.RC`'.

Conclusione del funzionamento del sistema NOS

Dal momento che il sistema NOS si comporta come una shell, si può intuire il modo attraverso il quale si conclude il suo funzionamento: con il comando '**`exit`**':

`net> exit` [*Invio*]

Guida interna

Come suggerisce lo stesso NOS quando si inserisce un comando errato, è disponibile una mini guida interna costituita dall'elenco dei comandi. Si ottiene con '**`help`**', oppure semplicemente con '**`?`**'. Non è molto, dal momento che non viene mostrata la sintassi rispettiva, comunque è sempre meglio di nulla.

`net> help` [*Invio*]

Interfacce, instradamento e nomi

Le cose più importanti da fare per poter utilizzare il sistema NOS, sono la definizione delle interfacce, l'instradamento e la risoluzione dei nomi. Le interfacce vengono «attaccate» attraverso il comando '**`attach`**', quindi vengono configurate attraverso '**`ifconfig`**', alla fine l'instradamento viene definito attraverso il comando '**`route`**'. NOS non ha funzionalità di DNS, a parte la possibilità di risolvere alcuni nomi a dominio per conto proprio, ma si può avvalere di un DNS esterno attraverso il comando '**`domain`**'.

I comandi che vengono descritti in queste sezioni sono usati generalmente per la configurazione attraverso il file '`AUTOEXEC.NET`'. Ciò dovrebbe essere intuitivo dato il tipo di operazioni che si svolgono con questi.

Comandi «attach» e «detach»

Attraverso il comando '**`attach`**' si possono definire le interfacce utilizzate. Di solito l'eseguibile '**`NET.EXE`**' è predisposto per la gestione delle porte seriali ('**`asy`**') e per l'uso di un driver di pacchetto esterno.

```
attach asy i/o irq { ppp | slip } nome_interfaccia dim_buffer mtu bps [c][r][v]
```

```
attach packet irq nome_interfaccia coda_trasmissione mtu
```

Quello che si vede rappresenta la sintassi per la definizione di un'interfaccia seriale (PPP, SLIP, o altre che non sono state indicate) e per un'interfaccia comandata da un driver di pacchetto esterno.

Nel caso del tipo '**`asy`**', cioè della connessione seriale, il numero di IRQ e l'indirizzo di I/O si riferiscono a quelli della porta seriale stessa; inoltre, gli ultimi argomenti sono la velocità espressa in bit/s (bps) e una stringa facoltativa dove possono apparire le lettere '**`c`**', '**`r`**' e '**`v`**'. Queste lettere rappresentano tre modalità: se appare la '**`c`**' si utilizza il protocollo RTS/CTS; se appare la '**`r`**' si abilita la sensibilità al segnale CD (*Carrier detect*); se appare la '**`v`**' si abilita la compressione Van Jacobson delle intestazioni TCP/IP, ma solo per le connessioni SLIP.

Con le interfacce gestite da un driver di pacchetto esterno diventa tutto più facile, dal momento che la cosa più importante è solo l'indicazione dell'indirizzo IRQ software (quello che serve a individuare il driver).

Per eliminare un'interfaccia si utilizza invece il comando '**`detach`**' secondo la sintassi seguente:

```
detach interfaccia
```

Segue la descrizione di alcuni esempi.

- `attach packet 0x7e ethernet0 8 1500`

  Utilizza un driver di pacchetto per gestire una scheda Ethernet. L'indirizzo IRQ per comunicare con il driver è $7E_{16}$; viene definito il nome '**`ethernet0`**' per fare riferimento a questa scheda; si pone il limite di otto pacchetti per la coda di trasmissione; si stabilisce l'unità massima di trasmissione in 1500 byte.

- `attach asy 0x3f8 4 slip sl0 1024 256 9600`

  Questo esempio è tratto dalla documentazione di NOS e si riferisce a una connessione SLIP attraverso la porta seriale individuata dall'indirizzo di I/O $3F8_{16}$ e dall'indirizzo IRQ 4. Il nome che viene attribuito è '**`sl0`**'; viene definito un *buffer* di ricezione di 1024 byte; la dimensione massima dei pacchetti trasmessi è di 256 byte; la velocità della porta seriale è di 9600 bit/s.

- `attach asy 0x3f8 4 ppp pp0 4096 1500 9600 r`

  Anche questo esempio è tratto dalla documentazione di NOS e si riferisce a una connessione PPP attraverso la porta seriale individuata dall'indirizzo di I/O $3F8_{16}$ e dall'indirizzo IRQ 4. Il nome che viene attribuito è '**`pp0`**'; viene definito un *buffer* di ricezione di 4096 byte; la dimensione massima dei pacchetti trasmessi è di 1500 byte; la velocità della porta seriale è di 9600 bit/s; viene abilitato il controllo della linea CD del modem.

- `detach ethernet0`

  Elimina l'interfaccia '**`ethernet0`**'.

Comando «ifconfig»

Attraverso il comando '**`ifconfig`**' si possono configurare le interfacce definite in precedenza con il comando '**`attach`**'. Il comando può assumere diverse forme, ma in particolare sono importanti gli schemi seguenti:

```
ifconfig [nome_interfaccia]
```

```
ifconfig nome_interfaccia ipaddress indirizzo_ip
```

```
ifconfig nome_interfaccia netmask maschera_ip
```

Utilizzando il comando da solo, senza argomenti, si ottiene la visualizzazione dello stato di tutte le interfacce di rete, comprese quelle predefinite; se si specifica il nome di un'interfaccia, il risultato si limita allo stato di questa. La seconda e la terza modalità servono invece per abbinare un indirizzo IP e una maschera di rete all'interfaccia.

Segue la descrizione di alcuni esempi.

- `ifconfig ethernet0`

  Mostra lo stato dell'interfaccia '**`ethernet0`**', che in precedenza è stata dichiarata con questo nome.

- `ifconfig ethernet0 ipaddress 192.168.1.10`

  Abbina all'interfaccia l'indirizzo IP 192.168.1.10.

- `ifconfig ethernet0 netmask 255.255.255.0`

  Abbina all'interfaccia la maschera IP 255.255.255.0.

Comando «route»

Il comando '**`route`**' permette di definire gli instradamenti attraverso le interfacce di rete configurate precedentemente, specificando eventualmente anche i router necessari a raggiungere le reti esterne.

```
route
```

```
route add indirizzo_ip/n_bit_maschera nome_interfaccia [router]
```

```
route add default nome_interfaccia [router]
```

La sintassi mostrata rappresenta una semplificazione del comando necessario a definire un instradamento. La coppia ***indirizzo_ip*/*n_bit_maschera*** è un modo per rappresentare l'indirizzo di una rete in modo compatto: il numero di bit rappresenta quanti bit iniziali devono essere posti a uno nella maschera di rete.

Per eliminare un instradamento si utilizza la forma seguente:

```
route drop indirizzo_ip/n_bit_maschera
```

```
route drop default
```

Segue la descrizione di alcuni esempi.

- `route`

  Mostra l'instradamento delle interfacce.

  `route add 192.168.1.0/24 ethernet0`

- Definisce l'instradamento per la rete identificata dagli indirizzi 192.168.1.∗ attraverso l'interfaccia '**`ethernet0`**'.

  `route add default ethernet0 192.168.1.254`

  Definisce l'instradamento predefinito attraverso il router 192.168.1.254.

Comando «domain»

Il comando '**`domain`**' permette di definire quali sono i servizi DNS a cui il sistema NOS può rivolgersi; permette anche di configurare il loro utilizzo e di definire eventualmente una risoluzione locale per alcuni indirizzi. Qui viene mostrato solo come dichiarare l'uso dei servizi DNS e il dominio predefinito.

```
domain addserver indirizzo_ip_DNS
```

```
domain suffix [suffisso_predefinito]
```

Segue la descrizione di alcuni esempi.

- ```
  domain addserver 192.168.1.1
  domain addserver 192.168.1.2
  ```

  Dichiara l'uso del servizio DNS collocato presso i nodi 192.168.1.1 e 192.168.1.2.

- `domain suffix brot.dg.`

- Dichiara che in caso di nomi a dominio incompleti viene aggiunto il suffisso *brot.dg*.

  `domain suffix`

  Mostra il suffisso predefinito per i nomi a dominio.

Gestione delle sessioni

Il sistema NOS può gestire diverse sessioni di lavoro, corrispondenti ad altrettante attività che implicano l'instaurarsi di una connessione. Per esempio si possono gestire diverse connessioni TELNET simultaneamente e lo stesso vale per l'utilizzo del protocollo FTP. Tutto questo funziona in modo paragonabile al sistema delle console virtuali di GNU/Linux: con il sistema NOS c'è una finestra per la modalità di comando, dove si trova l'invito, attraverso la quale si impartiscono i comandi, e le finestre delle sessioni che vengono aperte automaticamente in base al tipo di comando che viene dato.

Quando ci si trova a interagire con una sessione è possibile tornare alla finestra della modalità di comando attraverso il tasto [*F10*] (vale solo per il NOS che si basa sul Dos) e poi, da lì è possibile tornare a una sessione attraverso il comando '**`session`**'. Da questo si comprende che le sessioni sono numerate, cosa che avviene in modo automatico. Una di queste è anche la sessione attiva, ovvero quella a cui si potrebbe fare riferimento quando non se ne specifica il numero.

Comando	Descrizione
`session` [*n_sessione*]	Il comando '**`session`**' permette di tornare a una sessione ancora attiva, specificandone il numero. Se questo non viene indicato, si ottiene l'elenco delle sessioni esistenti.
`session`	Elenca le sessioni esistenti.
`session 1`	Torna alla prima sessione.
`close` [*n_sessione*]	Il comando '**`close`**' interrompere una connessione TCP attraverso l'invio di un pacchetto FIN (che serve a chiudere una connessione del genere). Il risultato che si ottiene di solito è che la sessione corrispondente termina.
`close`	Interrompe la connessione della sessione corrente.
`close 1`	Interrompe la connessione della prima sessione.
`reset` [*n_sessione*]	Il comando '**`reset`**' interrompere una connessione TCP attraverso l'invio di un pacchetto RST. Il risultato che si vuole ottenere è la conclusione della sessione corrispondente, ma dal momento che il metodo dell'invio di un pacchetto RST non garantisce l'ottenimento di ciò, sarebbe preferibile utilizzare il comando '**`close`**' al suo posto.
`abort` [*n_sessione*]	Il comando '**`abort`**' permette di interrompere un'operazione di carico o scarico dati attraverso una sessione FTP. La sessione in questione non viene chiusa.<
`abort`	Interrompe le operazioni di carico-scarico nella sessione corrente, purché di FTP.
`abort 1`	Interrompe le operazioni di carico-scarico nella prima sessione, purché sia di FTP.

Attività nel sistema locale

Dal momento che non è possibile intervenire direttamente sul sistema operativo sottostante senza interrompere le connessioni che eventualmente fossero state instaurate, NOS deve incorporare alcune funzionalità che non hanno attinenza con la rete, ma che sono indispensabili a livello pratico.

Per quanto riguarda i percorsi delle directory possono essere indicati utilizzando sia le barre oblique inverse ('\') che quelle normali ('**`/`**') per la separazione dei nomi che li compongono.

Comando	Descrizione
`cd` [*percorso*]	Permette di cambiare la directory corrente nel sistema sottostante. Se viene utilizzato senza l'indicazione del percorso, si ottiene la visualizzazione della directory corrente.
`pwd`	Mostra quale sia la directory corrente.
  ```

Comando	Descrizione
`dir` [*percorso*]	Elenca il contenuto della directory corrente.
`mkdir` [*percorso*]	Crea una directory nel sistema operativo sottostante.
`rmdir` [*percorso*]	Elimina una directory nel sistema operativo sottostante.
`more` *file*…	Scorre il testo di uno o più file del sistema operativo sottostante, facendo una pausa tra le schermate.
`rename` *file_origine file_destinazione*	Rinomina o sposta un file del sistema operativo sottostante.
`delete` *file*…	Elimina i file indicati negli argomenti dal sistema operativo sottostante.
`!` | `shell`	Sospende il funzionamento di NOS per aprire una shell del sistema operativo sottostante (‘**`COMMAND.COM`**’).

Gestione della rete e delle connessioni

Oltre a quanto visto inizialmente per ciò che riguarda la definizione delle interfacce, la loro configurazione, l’instradamento e la risoluzione dei nomi, c’è una serie di comandi e di funzionalità per la gestione della rete.

Indirizzi IP e indirizzi MAC

```
arp
```

```
arp flush
```

Il comando ‘**`arp`**’ permette di conoscere il contenuto della tabella di trasformazione degli indirizzi IP in indirizzi fisici e viceversa. Questa viene costruita automaticamente dal sistema, durante il suo funzionamento. Sono disponibili degli argomenti particolari per inserire a forza delle voci nella tabella, anche se questa operazione non dovrebbe essere necessaria. In particolare, il comando ‘**`arp flush`**’ svuota la tabella attuale, costringendo il sistema NOS a ricominciare a costruirsela.

Ping e instradamento

Attraverso il comando ‘**`ping`**’ si può inviare una richiesta di eco utilizzando il protocollo ICMP. Questo è il modo consueto per verificare che sia presente un certo nodo nella rete. In generale conviene utilizzare soltanto la sintassi seguente, con la quale viene inviata un’unica richiesta.

```
ping nodo
```

Per verificare il percorso dei pacchetti lungo la rete si può utilizzare il comando ‘**`hop`**’. Il comando normale si articola nel modo seguente:

```
hop check nodo
```

Tuttavia, si può intervenire su alcuni parametri di funzionamento di questo comando: il TTL (*Time to live*),

```
hop maxttl max_salti
```

l’attesa massima,

```
hop maxwait n_secondi
```

e il numero di pacchetti di prova che vengono inviati a ogni nodo.

```
hop queries n_pacchetti
```

Infine, è possibile abilitare o meno la visualizzazione di informazioni aggiuntive:


```
hop trace [on|off]
```

Varie

```
hostname [nome]
```

Attraverso il comando ‘**`hostname`**’ è possibile definire o visualizzare il nome attribuito al nodo. Questo dovrebbe corrispondere alla parte finale del nome a dominio, ma in ogni caso serve solo nei messaggi di presentazione del sistema.

```
socket [n_porta]
```

Attraverso il comando ‘**`socket`**’ è possibile conoscere lo stato delle porte. Utilizzandolo senza argomenti si ottiene l’elenco delle porte utilizzate, generalmente quelle dei servizi in ascolto ed eventualmente anche quelle gestite dalle sessioni in cui si utilizzano dei clienti di qualche tipo, mentre specificando una porta precisa si ottengono le statistiche sul traffico intrattenuto.

NOS come cliente

L’uso più importante del sistema NOS è quello di cliente in grado di utilizzare i servizi fondamentali di una rete TCP/IP. Si tratta principalmente di TELNET e FTP.

Cliente TELNET

Il sistema NOS permette di attivare una sessione TELNET verso un altro sistema che offra la possibilità di accedere attraverso questo tipo di protocollo. Purtroppo, il tipo di terminale corrispondente alla sessione TELNET è molto modesto, tanto che nelle versioni più limitate di NOS non si possono usare nemmeno i tasti freccia.

```
telnet nodo
```

Quando si utilizza questo tipo di cliente TELNET per accedere a un nodo corrispondente a un elaboratore GNU/Linux, il tipo di terminale che si vede nella variabile ‘**`TERM`**’ è ‘**`network`**’, che però non corrisponde ad alcuna voce nel sistema Terminfo o nel sistema Termcap. Eventualmente si può cambiare questo nome con ‘**`ansi`**’, o ‘**`ansi-mono`**’ se si preferisce.

Da una sessione TELNET è possibile tornare alla modalità di comando premendo il tasto [*F10*]. Per ritornare alla sessione con TELNET, si può poi utilizzare il comando ‘**`session`**’.

Cliente FTP

Il sistema NOS permette di attivare una sessione FTP. Una volta avviata, si ha a disposizione un cliente FTP tradizionale, con comandi molto simili a quelli del programma ‘**`ftp`**’ dei sistemi Unix.

```
ftp nodo
```

L’unico vero difetto sta nel sistema operativo sottostante: utilizzando il Dos i nomi dei file che vengono salvati sono ridotti al modello «8.3».

È importante ricordare di modificare sempre il tipo di trasferimento dati, in modo che sia di tipo binario (*image*): ‘**`type i`**’.

NOS come servente

I servizi offerti da NOS sono limitati e comunque dipendono dalla versione di questo sistema. Questi servizi devono essere abilitati attraverso il comando '**start**'. Dal momento che dipende dalla versione di NOS se un tipo di servizio è disponibile o meno, attraverso il comando '**start ?**' si ottiene l'elenco di questi.

In generale non conviene avere grandi pretese; probabilmente è il caso di attivare sempre i servizi '**discard**', '**echo**', '**ftp**' e '**finger**' (ammesso che questo ultimo possa avere senso).

```
start discard
start echo
start finger
start ftp
```

Per converso, volendo disattivare un servizio basta utilizzare il comando '**stop**' nello stesso modo.

Registrazione degli eventi

NOS permette di annotare gli accessi in un registro abbastanza semplificato. Si attiva questa funzionalità attraverso il comando '**log**':

```
log [stop | file_delle_registrazioni]
```

Per esempio, per attivare la registrazione degli accessi nel file '`C:\ACCESSI.LOG`', si può usare il comando seguente:

```
log c:\accessi.log
```

Come si può intuire, il comando '**log stop**' termina l'attività di registrazione degli accessi, senza interferire con gli accessi stessi. Infine, il comando '**log**' senza argomenti permette di sapere se questo sia attivo e in tal caso su quale file vengono fatte le annotazioni.

Servente FTP

Per abilitare il servizio FTP, oltre che usare il comando '**start ftp**', occorre predisporre un file di autorizzazioni: '`ftpusers`' collocato nella directory radice del servizio NOS. Il file deve contenere delle righe scomposte in quattro campi separati da uno o più spazi e si possono indicare anche dei commenti che si introducono con il simbolo '`#`'.

```
utente  parola_d'ordine  percorso  permessi
```

I quattro campi sono obbligatori e il significato è intuitivo:

1. ***utente*** serve a specificare il nome dell'utente che può accedere;
2. ***parola_d'ordine*** rappresenta la parola d'ordine in chiaro necessaria per l'accesso -- se si utilizza un asterisco ('`*`'), viene accettata qualunque parola d'ordine;
3. ***percorso*** indica la directory a partire dalla quale si concede l'accesso all'utente;
4. ***permessi*** è un numero che esprime i permessi consentiti all'utente.

I permessi non sono indicati secondo la tradizione Unix, quindi occorre fare attenzione. I permessi sono espressi con un solo numero ottenuto sommandone altri, che comunque si riferiscono alla directory di partenza e a tutte le sottodirectory: uno rappresenta un permesso di lettura; due rappresenta un permesso di creazione (di aggiunta di file senza poter sovrascrivere o eliminare quelli esistenti); quattro rappresenta un permesso di scrittura (o di sovrascrittura). Si osservi l'esempio seguente:

```
tizio tazza \home\tizio 7
caio capperi \home\caio 7
sempronio sempre \progetto 3
ftp * \pub 1
anonymous * \pub 1
```

Gli utenti '**tizio**' e '**caio**' hanno una loro directory personale in cui possono fare quello che vogliono; l'utente '**sempronio**' partecipa a un lavoro che si trova nella directory '`\PROGETTO\`' e lì ha la possibilità di immettere file, senza cancellare o sovrascrivere quelli presenti. Infine, gli utenti '**ftp**' e '**anonymous**' accedono con una parola d'ordine qualunque alla directory '`\PUB\`', con il solo permesso di lettura.

NOS come router IPv4

NOS funziona perfettamente come router se l'elaboratore in cui si utilizza dispone di più interfacce di rete. A titolo di esempio viene mostrato in che modo potrebbero essere utilizzate due schede di rete compatibili NE2000. Supponendo che queste utilizzino rispettivamente le risorse IRQ 10, I/O 280_{16}, e IRQ 11, I/O 300_{16}, la configurazione del driver di pacchetto (si fa riferimento a quanto descritto nella sezione u0.1) potrebbe essere quella seguente:

```
NE2000 0x60 0x0a 0x280
NE2000 0x61 0x0b 0x300
```

Supponendo che queste due schede servano a connettere le reti 192.168.1.∗ e 192.168.2.∗, supponendo anche che l'instradamento predefinito passi per il router 192.168.1.254, il file di configurazione di NOS potrebbe contenere in particolare le righe seguenti:

```
# Configurazione delle interfacce di rete.
attach packet 0x60 ethernet0 8 1500
attach packet 0x61 ethernet1 8 1500

# Definizione degli indirizzi IP.
ifconfig ethernet0 ipaddress 192.168.1.10
ifconfig ethernet0 netmask 255.255.255.0
ifconfig ethernet1 ipaddress 192.168.2.10
ifconfig ethernet1 netmask 255.255.255.0

# Instradamento.
route add 192.168.1.0/24 ethernet0
route add 192.168.2.0/24 ethernet1
route add default ethernet0 192.168.1.254
```

Non c'è bisogno di fare altro: l'attraversamento dei pacchetti da un'interfaccia all'altra avviene automaticamente (purché gli instradamenti siano corretti).

[1] **NOS** GNU GPL come descritto in *http://www.ka9q.net/code/*

[2] NOS è disponibile in varie versioni per diversi sistemi operativi: PMNOS per Presentation Manager (OS/2), AmigaNOS per Amiga e TNOS per GNU/Linux! L'attenzione di questo capitolo è comunque rivolta alle versioni di NOS per Dos.

nanoDos

Organizzazione della distribuzione 387
Installazione 387
Configurazione 388
Utilizzo 389
Conclusione 389
Riferimenti 389

nanoDos è una sorta di distribuzione FreeDOS (per architettura x86-16), finalizzata all'utilizzo di qualche servizio di rete essenziale, basata sull'uso delle librerie WATTCP.

Il software contenuto nella distribuzione ha licenze di vario tipo; in ogni caso si intende che il software sia utilizzabile gratuitamente senza limitazioni. Il lavoro di realizzazione della raccolta nanoDos è semplicemente di pubblico dominio e offerto senza alcuna garanzia e senza sostegno di alcun tipo.

Organizzazione della distribuzione

La distribuzione è composta da un file-immagine, da usare per riprodurre il dischetto di avvio, e da una serie di file con estensione '`.zip`', da estrarre successivamente all'installazione del contenuto del dischetto di avvio.

Il dischetto di avvio che si ottiene dal file-immagine appena descritto, può essere usato così come si trova, senza installarlo, ma in tal caso è disponibile solo il sistema operativo e un programma per l'accesso a un servente TELNET.

Per la precisione sono disponibili due file-immagine: uno adatto per dischetti da 1440 Kibyte (da 90 mm, ovvero da 3,5 in) e uno per i vecchissimi dischetti flessibili da 1200 Kibyte (da 133,35 mm, ovvero da 5,25 in). I nomi dei file sono rispettivamente '`boot.144`' e '`boot.120`'. Tanto per richiamare la memoria, in un sistema GNU/Linux è possibile riprodurre il dischetto con uno dei due comandi seguenti:

```
# cp boot.144 /dev/fd0 [Invio]
```

```
# cp boot.120 /dev/fd0 [Invio]
```

I file con estensione '`.zip`' contengono degli applicativi, suddivisi in modo da occupare ognuno una directory separata. Questi file possono essere usati dopo che il contenuto del dischetto di avvio è già stato installato nel disco fisso, per installare le applicazioni rimanenti.

Installazione

Una volta realizzato il dischetto di avvio adatto al proprio elaboratore, lo si può utilizzare per l'avvio del sistema operativo, con il quale si può procedere a predisporre la partizione del disco fisso che deve accoglierlo. Il dischetto **non** deve essere protetto contro la scrittura, perché durante il funzionamento vengono creati dei file temporanei.

Il sistema avviato da dischetto si presenta nello stesso modo in cui si presenterebbe una volta installato nel disco fisso, mostrando un menù di funzioni disponibili, che in realtà sono quasi tutte assenti.

Una volta avviato il dischetto, se necessario, si può usare il programma '**`FDISK.EXE`**' per creare le partizioni; successivamente occorre inizializzare la partizione che deve accogliere il sistema con il programma '**`FORMAT.EXE`**':

```
A\> FORMAT C: /S [Invio]
```

Fatto questo, si passa alla copia del dischetto, tale e quale, nel disco ‘C:’:

```
A\> XCOPY A:\*.* C:\*.* /E /S /H /V [Invio]
```

Se tutto va bene, si può riavviare e vedere che il sistema parte regolarmente dal disco fisso. Se le cose stanno così, si possono copiare i file ‘.zip’ in alcuni dischetti, per estrarli successivamente con un comando del genere:

```
C\> UNZIP A:\*.ZIP [Invio]
```

Per ogni file ‘.zip’, si deve ottenere una directory corrispondente che parte dalla radice del disco fisso, con lo stesso nome, ma senza l’estensione ‘.zip’.

Se tutto questo funziona con successo, il sistema nanoDos è pronto.

Configurazione

Per configurare nanoDos è necessario modificare il file ‘\AUTOEXEC.BAT’.

Inizialmente vengono definite alcune variabili di ambiente che contengono le informazioni necessarie alla gestione del TCP/IP e dell’interfaccia di rete:

```
…
set MY_IP=172.21.4.1
set NETMASK=255.255.0.0
set GATEWAY=172.21.254.254
set NAMESERVER=172.21.254.254
set HOSTNAME=nanodos

set NIC_DRIVER=\crynwr\ne2000.com
set NIC_IRQ_SOFT=0x7e
set NIC_IRQ=0x0b
set NIC_IO=0x300
…
```

Come si intende, le variabili di ambiente con il nome che corrisponde al modello ‘**NIC**_∗’ servono alla configurazione relativa all’interfaccia di rete. In condizioni normali è previsto l’uso di una scheda NE2000, attraverso il programma ‘\CRYNWR\NE2000.COM’, configurata in modo da usare l’indirizzo IRQ 11 ($0B_{16}$) e l’indirizzo I/O 300_{16}; inoltre viene specificato l’uso dell’indirizzo IRQ $7E_{16}$ per la comunicazione con le applicazioni. Naturalmente bisogna conoscere come è configurata la propria scheda e se necessario si devono usare i programmi appropriati per configurarla come si desidera.

Le variabili di ambiente precedenti a queste dichiarano l’indirizzo IP dell’elaboratore, la maschera di rete, il router per le altre reti, il servente per la risoluzione dei nomi e il nome dell’elaboratore stesso. Più avanti nel file ‘\AUTOEXEC.BAT’ queste variabili vengono usate per costruire i vari file di configurazione necessari ai programmi che usano la rete.

Tabella u62.2. Variabili di ambiente usate per la configurazione della rete.

Variabile	Esempio	Descrizione
MY_IP	172.21.4.1	Indirizzo IPv4 dell’elaboratore.
NETMASK	255.255.0.0	Maschera di rete.
GATEWAY	172.21.254.254	Router per le altre reti.
NAMESERVER	172.21.254.254	Servente DNS.
HOSTNAME	nanodos	Nome dell’elaboratore.
NIC_DRIVER	‘\CRYNWR\NE2000.COM’	Programma per la gestione della scheda di rete.
NIC_IRQ_SOFT	$7E_{16}$, 126_{10}	Indirizzo IRQ usato per comunicare con le applicazioni (rimane fisso).
NIC_IRQ	$0B_{16}$, 11_{10}	Indirizzo IRQ dell’interfaccia di rete (dipende dalla configurazione dell’interfaccia).
NIC_IO	300_{16}, 768_{10}	Indirizzo I/O dell’interfaccia di rete (dipende dalla configurazione dell’interfaccia).

Nel file ‘\AUTOEXEC.BAT’ si può intervenire anche per modificare la mappa della tastiera, che secondo la configurazione predefinita è quella italiana; diversamente si può usare a mano il programma ‘**KEYB.EXE**’:

```
…
\freedos\keyb it,850,\freedos\key\it.kl
…
```

Eventualmente, c’è da considerare la configurazione del mouse, che inizialmente è predisposta per un mouse PS/2. Per intervenire in questo occorre modificare il file ‘\MOUSE.BAT’, eventualmente così per usare un mouse seriale collegato alla prima porta (‘COM1:’):

```
REM Serial mouse at COM1:
\driver\ctmouse.exe /S1
REM PS2 mouse
REM \driver\ctmouse.exe /P
```

Se poi si vuole utilizzare un elaboratore i386 o superiore, può essere conveniente modificare anche il file ‘\CONFIG.SYS’, in modo da abilitare l’uso di ‘HIMEM.SYS’:

```
files=100
buffers=25
dos=HIGH,UMB
REM country=039,,\freedos\country.sys
lastdrive=z
SHELL=COMMAND.COM /E:2048 /P
STACKS=9,256
device=\driver\himem.exe
REM device=\driver\emm386.exe
```

Si osservi che queste modifiche vanno apportate anche sul dischetto, se si intende usare il sistema minimo senza installarlo nel disco fisso.

Utilizzo

Una volta avviato il sistema operativo, dopo una breve schermata di avvertimento sulla mancanza assoluta di garanzie, appare un menù che riepiloga l’uso dei programmi principali.

Figura u62.6. Menù di nanoDos.

```
.-------------------------------------------------------------------------------.
|LYNX host                  |Simple HTTP navigator.                             |
|ARACHNE http://host/...    |Graphic HTTP navigator for i386.                   |
|SSH user host              |SSH 1 client.                                      |
|TELNET host                |TELNET client.                                     |
|FTP host                   |FTP client.                                        |
|TALK user@host             |Call a user at a specified host.                   |
|TALK -a                    |Ready to receive a TALK call.                      |
|PING host                  |Ping to host.                                      |
|NSLOOKUP domain            |Domain resolution.                                 |
|NSQUERY -d domain          |Domain resolution with more infos.                 |
|TROUTE host                |Trace the route to host.                           |
|FINGER [user]@host         |FINGER client.                                     |
|LPR [queue] host file      |Print the local file to the remote printer.        |
|LPQ -Pqueue -Shost         |Query remote printer queue.                        |
|TCPINFO                    |See network configuration.                         |
|---------------------------|---------------------------------------------------|
|MENU                       |Recall this menu.                                  |
|---------------------------|---------------------------------------------------|
|[Ctrl]+[Alt]+[Del]         |Restart the computer.                              |
`-------------------------------------------------------------------------------'
```

Si osservi che Arachne funziona soltanto in un sistema i386 o superiore, dove deve essere stato abilitato l’uso di ‘HIMEM.SYS’; inoltre non è molto stabile.

Conclusione

nanoDos è un sistema abbastanza fragile, con il quale ogni tanto si può essere costretti a riavviare l’elaboratore, ma può consentire l’uso efficace di vecchie macchine, sia a scopo didattico, sia a scopo professionale, se ciò che serve è un terminale senza pretese.

Il senso di questo lavoro sta soprattutto nell’organizzazione della configurazione. Da quanto fatto, se si conosce il sistema operativo, è facile realizzare una riduzione che si adatti meglio alle proprie esigenze.

Riferimenti

- *nanoDos*
  extra/nanoDos/

Parte xxii

ALML

Introduzione ad ALML 395
- Esigenze che Alml intenderebbe soddisfare 395
- Tutto in uno 395
- Continuità tra documento stampato e documento elettronico 395
- Perché SGML 396
- Capitoli sullo stile usato per a2 396

Preparazione e visione generale 397
- Installazione di Alml 397
- Esempio iniziale 398
- Cosa si genera con la composizione 401
- Sintassi nell'uso del programma frontale 402
- Codifica del sorgente 406
- Organizzare un file-make o uno script 406
- Formati particolari 408
- Progetti di documentazione che utilizzano il formato di Alml 409

Il documento secondo Alml 411
- Organizzazione generale 411
- Dalla copertina all'indice generale 412
- Contenuto 419
- Documento multilingua 425
- Definizione alternativa della suddivisione del documento 426

Elementi interni alle righe 429
- Numeri 430
- Tastiera, menù e codice ASCII 431
- Indirizzi di posta elettronica 432

Blocchi comuni 433
- Elenchi e simili 433
- Testo letterale o quasi 435
- Modelli sintattici 439
- Comandi 441

Altri blocchi e componenti lineari particolari 443
- Inserzioni particolari 443
- Riquadri 444
- Copia di porzioni del documento 446

Riferimenti, note e altre informazioni 449
- Riferimenti incrociati e ipertestuali 449
- Note e piè pagina 451
- Riferimenti esterni e citazioni 451
- Indici analitici e termini speciali 452
- Caratteristiche del software e di altri «lavori» 455
- Informazioni su sezioni specifiche del documento 456
- Sezioni particolari 457

Immagini e video 459
- Immagini esterne 462
- Immagini incorporate Base64 462
- Immagini incorporate EPS 463
- Immagini incorporate XFig 463
- Immagini incorporate LilyPond 464
- Immagini incorporate TeX e LaTeX 464
- Immagini incorporate Gnuplot 465

Osservazioni sull'incorporazione di codice estraneo 466
Tabelle 467
Allegati 471
Verifiche 473
Capitolo per le verifiche 474
Impedire la lettura del codice 476
Esempio di verifica 476
Esempio di verifica con Alml 481
Esempio di verifica con Alml bis 483
Esempio di verifica con Alml ter 485
Presentazioni 487
Esempio di presentazione 487
Composizione 488
Inserimento letterale di codice TeX e HTML, con eventuale inserimento condizionato 491
Entità ISO ed entità HTML gestite da Alml 493
Alfabeti simbolici 493
Alfabeti latini 502
Alfabeti non latini 505
HTML 508
Riferimenti 512
Insieme di caratteri universale e Alml 513
Riferimenti 535
Stile di scrittura del sorgente 537
Blocchi di testo e rientri 537
Figure e tabelle 538
Titoli 539
Sezioni marcate 539
Alml per i grandi progetti di documentazione 541
Estrapolazione di porzioni del file SGML 541
Esempio di un progetto 542
Aggregazioni 543
Questioni tecniche 545
Usare Textchk e Ispell con Alml 545
Espandere le potenzialità elaborative di TeX 545
Programma di supporto 548
Gestione di «a2» 555
Articolazione dei file del sorgente 555
Inclusione selettiva dei file esterni ed entità speciali 555
Composizione guidata con il file-make 557
Convenzioni di «a2» 559
Unità di misura e moltiplicatori 559
Casi particolari di testo che non viene enfatizzato 560
Valori numerici in lettere e in cifre 560
Distinzione nell'uso dei nomi degli applicativi 561
Descrizione degli acronimi 562
Indice analitico 562
Enfatizzazioni e uso degli elementi «special» 564
Rappresentazione del contenuto di file e dei flussi standard 568
Altri problemi di coerenza nell'uso degli elementi SGML . 569
Sezioni marcate per le annotazioni 570
Glossario stilistico di «a2» 571
Termini tecnici particolari 572

Glossario 575
Forme espressive particolari 595
Annotazioni varie 595
Nomi dei caratteri speciali 596
Nomi da usare in modo uniforme 596
Riferimenti 597
Indice del glossario stilistico 597

Introduzione ad ALML

Esigenze che Alml intenderebbe soddisfare 395
Tutto in uno 395
Continuità tra documento stampato e documento elettronico . 395
Perché SGML 396
Capitoli sullo stile usato per a2 396

Alml[1] è un sistema di composizione SGML, realizzato espressamente per l'opera *a2*, ma che, in linea di principio, può andare bene per vari tipi di esigenze editoriali.

Esigenze che Alml intenderebbe soddisfare

Alml nasce e si sviluppa con l'obiettivo di consentire, in pratica, la realizzazione e la gestione di un documento con contenuti molto vari e di grandi dimensioni, quale può essere il già citato *a2*.

Esiste una grande varietà di strumenti per l'editoria elettronica, che però spesso hanno il difetto di essere troppo specifici. Per esempio, se con LaTeX (capitolo 50) si possono fare cose molto belle, bisogna considerare che non si può ottenere tutto assieme nello stesso momento, in quanto l'uso di certi stili condiziona il funzionamento di altri. Tuttavia, in generale questo problema non si avverte, perché di norma si realizzano documenti su un tema preciso, che richiede certe funzionalità e non altre.

Esiste anche una discreta quantità di strumenti generici, molto ben studiati per poter considerare «tutto» o quasi tutto, ma poi questi hanno il difetto di non avere ancora messo in pratica completamente quello che in teoria prevedono di poter fare.

Alml ha l'intento di essere uno strumento tipografico abbastanza generalizzato per poter scrivere di qualunque cosa, se, piuttosto di pretendere l'ottimo, ci si accontenta di risultati «decenti». Di conseguenza, l'obiettivo di Alml è quello di essere uno strumento alla portata di un singolo che vuole o che ha la necessità di gestire un lavoro variegato ed eventualmente di grandi dimensioni, con la ragionevole tranquillità di poter ottenere in pratica quasi tutto ciò che, in teoria, Alml promette di fare.

Tutto in uno

Una caratteristica significativa di Alml è quella di consentire, volendo, di mettere tutto in un solo file SGML, comprese le immagini e addirittura degli allegati, che potrebbero tradursi in file da scaricare durante la consultazione in linea. Ciò ha sicuramente lo svantaggio di far lavorare con un file gigantesco, ma ha il vantaggio di non fare perdere tempo nell'organizzazione e nella gestione di un insieme di file che può diventare troppo numeroso. Ciò è in pratica un punto a favore dell'utilizzo da parte di un singolo, che non abbia la necessità di avvalersi della collaborazione altrui. Naturalmente si può obbiettare che esiste il rischio di perdere tutti i dati più facilmente, ma in tal caso si parte dal presupposto che chi fa una cosa del genere, sappia anche premunirsi da incidenti di questo tipo.

Esiste comunque la possibilità di gestire con Alml un progetto composto da più file, così come in generale consente un sistema SGML comune, ma attraverso programmi accessori ad Alml è possibile anche organizzare un'aggregazione di più documenti autonomi, come è possibile l'estrapolazione di una porzione più piccola da uno o più documenti.

Continuità tra documento stampato e documento elettronico

Alml, che è un sistema SGML, ha una vaga somiglianza con HTML, ma non dà la stessa libertà, per garantire la produzione di formati finali differenti, ma coerenti tra di loro. Alml deve poter produrre, principalmente, un documento stampabile o un documento elettronico adatto alla consultazione in linea.

A titolo di esempio, si può considerare il caso dei riferimenti ipertestuali, che devono avere un senso, sia quando il documento viene stampato su carta, sia quando il documento viene letto in forma elettronica. In questo caso, Alml impone che il riferimento sia ben visibile in ogni circostanza, mentre usando HTML, i riferimenti potrebbero essere resi invisibili.

Un altro esempio più importante è dato dai capitoli speciali per la realizzazione di questionari di valutazione, che possono essere resi sia come documento stampato (e quindi statico), sia come documento interattivo, in grado di generare anche la valutazione in modo automatico (tramite JavaScript).

Perché SGML

Il sistema tipografico universale e libero del futuro sarà basato probabilmente su un linguaggio XML, ma lo scopo pratico di Alml non richiede le funzionalità di XML e, d'altro canto, può sfruttare funzionalità di SGML che invece XML ha abbandonato: le sezioni marcate (sezione 51.1.7). A ogni modo, la dichiarazione SGML di Alml incorpora alcune caratteristiche tipiche di un sistema XML, in modo particolare per ciò che riguarda la codifica universale dei caratteri.

Capitoli sullo stile usato per a2

Dopo i capitoli che descrivono il funzionamento di Alml, ne appaiono altri sullo stile di scrittura di a2. Quei capitoli sono solo indicativi e non sono aggiornati da diverso tempo; tuttavia rimangono assieme alla documentazione di Alml, per lasciare almeno un'idea di come è organizzata stilisticamente l'opera a2.

[1] **Alml** GNU GPL

Preparazione e visione generale

Installazione di Alml 397
Gettext 398
Esempio iniziale 398
Cosa si genera con la composizione 401
Sintassi nell'uso del programma frontale 402
Codifica del sorgente 406
Organizzare un file-make o uno script 406
Formati particolari 408
Progetti di documentazione che utilizzano il formato di Alml 409

Alml è costituito principalmente da un programma Perl ('**`alml`**') che controlla l'analizzatore SGML e altri programmi necessari per arrivare alla composizione finale del documento. Tuttavia, per poter comprendere tale meccanismo, sarebbe opportuno prima conoscere quanto descritto a proposito dell'SGML, di TeX e dei sistemi comuni di composizione basati su SGML.

Alml si avvale di altri programmi per l'analisi SGML e per la generazione di alcuni formati finali. In particolare, è necessario disporre di '**`nsgmls`**' che fa parte generalmente del pacchetto SP (anche se la propria distribuzione GNU potrebbe nominarlo in modo differente); inoltre è fondamentale la presenza di LaTeX per generare i formati da stampare. La tabella u64.1 riepiloga gli applicativi principali da cui dipende il buon funzionamento di Alml.

Tabella u64.1. Applicativi principali da cui dipende Alml.

Applicativo	Compito
Perl	Alml è scritto in Perl.
Perl-gettext	Modulo Perl per l'utilizzo di Gettext.
SP	Verifica la validità SGML e genera una prima conversione.
distribuzione TeX	Sistema di composizione che comprende TeX, LaTeX e altri lavori derivati.
PSUtils	Riorganizza, ingrandisce e riduce un file PostScript.
Dvipdfm	Consente una conversione in PDF a partire dal file DVI.
Uuencode	Estrae le immagini incorporate da file esterni.
GraphicsMagick o ImageMagick	Converte i file delle immagini nei formati appropriati, adattando le dimensioni.
Ghostscript	Serve a ImageMagick per la conversione di file PostScript in altri formati.
HTML2ps	Consente l'incorporazione di codice HTML nella composizione per la stampa.
W3M	Converte un file HTML in testo puro.
LilyPond	Consente l'incorporazione di codice LilyPond.
XFig	Consente l'incorporazione di codice XFig.
Gnuplot	Consente l'incorporazione di codice Gnuplot.
Eukleides	Consente l'incorporazione di codice Eukleides.
Groff, PS2EPS	Consentono l'incorporazione di codice *roff.
PlotUtils	Consente l'incorporazione di codice da vari programmi del pacchetto PlotUtils.

Installazione di Alml

Alml viene fornito attraverso archivi tradizionali di tipo tar+gzip, oppure in archivi Debian, in file con nomi del tipo:

```
alml-versione.tar.gz
```

```
alml_versione-n_all.deb
```

Estraendo il contenuto dell'archivio, si dovrebbero ottenere in particolare i file e le sottodirectory elencati nella tabella u64.2, che rappresentano l'essenziale.

Tabella u64.2. Contenuto essenziale dell'archivio di distribuzione di Alml.

File o directory	Descrizione
`bin/*`	File eseguibili.
`doc/*`	Esempi e documentazione eventuale.
`etc/*`	File di configurazione da inserire a partire dalla directory `/etc/`.
`man/*`	Pagine di manuale relative agli eseguibili.
`share/sgml/*`	File e directory da collocare in `/usr/share/sgml/alml/`.

Gli eseguibili che nel pacchetto di distribuzione si trovano nella directory `bin/`, devono essere raggiungibili attraverso il percorso di ricerca del sistema, rappresentato dalla variabile di ambiente '**`PATH`**'. Pertanto vanno collocati opportunamente, oppure vanno predisposti dei collegamenti adeguati.

Quanto contenuto nella directory `share/sgml/`, va collocato nella directory `/usr/share/sgml/alml/`, oppure vanno realizzati dei collegamenti equivalenti.

In generale, se la propria distribuzione GNU/Linux non è predisposta per la gestione delle entità standard ISO 8879, conviene modificare il collegamento simbolico '`alml.cat`', che nella sua collocazione finale deve trovarsi nella directory '`/usr/share/sgml/alml/`'. Normalmente questo punta al file '`alml.cat.debian`', ma in caso di problemi conviene modificarlo in modo che punti a '`alml.cat.normal`'.

Gettext

«

I messaggi di Alml possono essere tradotti. Se si dispone del file PO relativo alla lingua preferita, è necessario compilarlo come nell'esempio seguente:

```
$ msgfmt -vvvv -o alml.mo it.po [Invio]
```

In questo esempio, il file '`it.po`' viene compilato generando il file '`alml.mo`'. Trattandosi evidentemente della traduzione italiana, questo file può essere collocato in '`/usr/share/locale/it/LC_MESSAGES/`', o in un'altra posizione analoga in base agli standard del proprio sistema operativo.

Se non è disponibile il modulo Perl-gettext,[1] che serve a Alml per accedere alle traduzioni, è possibile eliminare il suo utilizzo e simulare la funzione di Gettext. In pratica si commentano le istruzioni seguenti all'inizio dei programmi '**`alml`**' e '**`alml-extra`**':

```
# We *don't* want to use gettext.
#use POSIX;
#use Locale::gettext;
#setlocale (LC_MESSAGES, "");
#textdomain ("alml");
```

Inoltre, si tolgono i commenti dalla dichiarazione della funzione fittizia ***gettext()***, come si vede qui:

```
sub gettext
{
    return $_[0];
}
```

Esempio iniziale

«

Un esempio iniziale può servire per comprendere il funzionamento generale di Alml (il file in questione dovrebbe essere disponibile presso *allegati/a2/alml-esempio-iniziale.sgml*). Il testo umoristico contenuto è di dominio pubblico.


```
<!DOCTYPE ALML PUBLIC "-//D.G.//DTD Alml//EN">
<alml lang="it" spacing="uniform">
<head>
    <admin>
        <description>Strafalcioni e sciocchezze varie</description>
        <keywords>strafalcione, svarione, detto, scherzo</keywords>
        <printedfontsize type="normal">7mm</printedfontsize>
    </admin>
    <title>Branchi di nebbia</title>
    <subtitle>I detti di oggi</subtitle>
    <author>Anonimo &lt;anonimo@brot.dg&gt;</author>
    <date>1111.11.11</date>
    <legal>
        <p>Il testo contenuto in questo documento è di dominio pubblico,
        pertanto ci si può fare quello che si vuole.</p>
    </legal>
    <maincontents levels="2">Indice generale</maincontents>
</head>
<intro>
<h1>
Introduzione al documento
</h1>

<p>Questo documento è scritto per dimostrare il funzionamento di Alml,
utilizzando frasi che, storpiando vecchi detti comuni, potrebbero
diventare i detti di domani.</p>

</intro>
<body>
<h1 id="capitolo-primo">
Attenzione ai branchi di nebbia... nella testa
<indexentry>nebbia</indexentry>
</h1>

<p>Sono scremato dalla fatica: il lavoro mobilita l'uomo, ma qui si
batte la fiaccola. Non fatemi uscire dai gamberi e stendiamo un velo
peloso: non bisogna foschilizzarsi così.</p>

<p>Durante le notti di pediluvio, arrivano certe zampate di caldo... C'è
il divieto di balenazione e all'improvviso arriva un'onda anonima:
bisogna fare attenzione ai branchi di nebbia.</p>

<h2>
Tappeti rullanti
<indexentry>metropolitana</indexentry>
<indexentry>treno</indexentry>
<indexentry>automobile</indexentry>
</h2>

<p>In metropolitana ci sono i tappeti rullanti, ma la domenica certi
treni vengono oppressi.</p>

<p>Una volta ho urtato la macchina sul paraguail, poi sono finito sulle
banchine spargitraffico e così ho perso la marmitta paralitica... Meno
male che l'auto aveva l'<em>iceberg</em> incorporato. Purtroppo, però,
mi hanno fatto la multa per guida in stato di brezza.</p>

<h1 id="capitolo-secondo">
Abete alto
<indexentry>dolce</indexentry>
<indexentry>sapone</indexentry>
</h1>

<p>Mi dispiace, non posso mangiare dolci, perché ho l'abete alto e
non posso permettermi neanche una zolla di zucchero nel caffé.</p>

<p>Sono pieno di malattie: ho le piastrelle basse; ho lo zagarolo
nell'occhio; ho una spalla lustrata; ho le vene vorticose... Ormai credo
di essere spizzotremito; mi hanno prescritto di fare i raggi
ultraviolenti.</p>

<p>Allora sono andato in farmacia per comprare il sapone clinicamente
intestato, ma poi ho preso del bicarbonato di soia e della tintura di
odio per combattere gli isterismi della cellulite.</p>

</body>
<appendix>
<h1>
Gondole voraci
</h1>

<p>Al ristorante ho ordinato un piatto di pasta con le gondole voraci,
una frittura di crampi, funghi traforati, un dolce con l'uva passera
ricoperto da zucchero al vento (cotto nel forno a microbombe), pesche
sciroccate, una birra doppio smalto e del latte pazzamente stremato.
Alla fine, mi sono fatto mettere gli avanzi nella carta spagnola.</p>

<p>Non mi voglio divulgare, ma di fronte a queste cose rimango
putrefatto... Così ho deciso che quando muoio mi faccio cromare.</p>

</appendix>
<index>
<h1>
Indice analitico
</h1>

<printindex index="main">

</index>
</alml>
```

Se tutto viene copiato correttamente nel file ipotetico '`alml-`

`esempio-iniziale.sgml`', con il comando seguente si ottiene la composizione in PostScript, attraverso LaTeX e Dvips:

```
$ alml --ps alml-esempio-iniziale.sgml [Invio]
```

Con il comando seguente, si ottiene la composizione in PDF, attraverso LaTeX e Dvipdfm:

```
$ alml --pdf alml-esempio-iniziale.sgml [Invio]
```

Con il comando seguente, si ottiene la composizione in HTML, su più file distinti:

```
$ alml --html alml-esempio-iniziale.sgml [Invio]
```

Il risultato che si dovrebbe ottenere, in formato PDF, può essere prelevato presso *allegati/a2/alml-esempio-iniziale.pdf* (viene distribuito assieme all'edizione HTML dell'opera).

Figura u64.6. Prima pagina (copertina) del risultato della composizione.

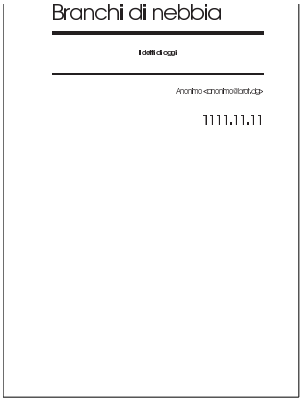


Figura u64.7. Seconda e terza pagina del risultato della composizione.

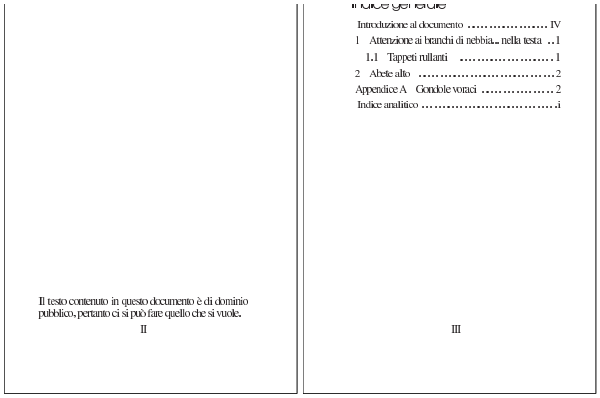


Figura u64.8. Quarta e quinta pagina del risultato della composizione.

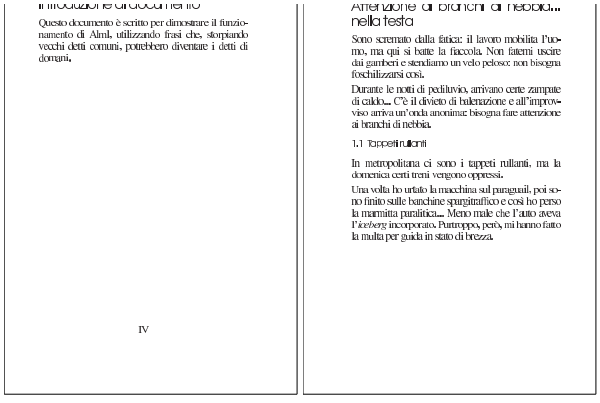


Figura u64.9. Sesta e settima pagina del risultato della composizione.

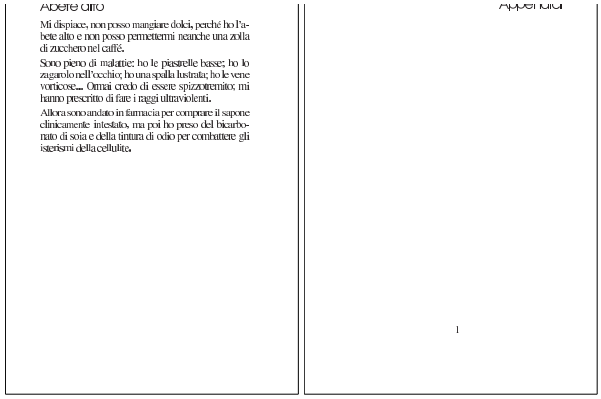
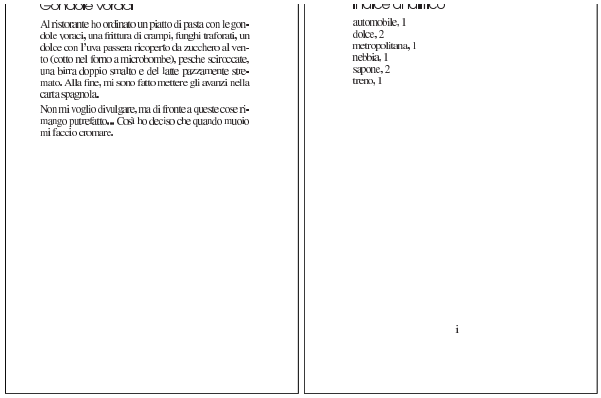


Figura u64.10. Ottava e nona pagina del risultato della composizione.



Cosa si genera con la composizione

L'utilizzo di Alml può generare file differenti a seconda del tipo di operazione che viene richiesta. La tabella u64.11 riepiloga i file principali.

Tabella u64.11. Alcuni file generati dall'utilizzo di Alml. Il file '***nome***`.sgml`' deve essere già presente.

File	Descrizione
'***nome***`.sgml`'	Il sorgente SGML principale da cui hanno origine gli altri file.
'***nome***`.aux`'	File ausiliario e temporaneo della composizione attraverso LaTeX.
'***nome***`.diag`'	File diagnostico generato da '**`alml`**'.
'***nome***`.pageref`'	File temporaneo con i riferimenti alle pagine nella composizione con LaTeX.
'***nome***`.pageloc`'	File contenente i riferimenti alle pagine per individuare i volumi e le parti, quando questi vanno estratti separatamente.
'***nome***`.log`'	File diagnostico generato da LaTeX.
'***nome***`.sp`'	File intermedio, ottenuto dall'elaborazione SGML di SP.
'***nome***`.sp2`'	File intermedio, ottenuto rielaborando il file '***nome***`.sp`', per sostituire le entità di tipo «SDATA» in codice appropriato per il tipo di composizione prescelto.
'***nome***`.dvi`'	Composizione in DVI, finale o transitoria.
'***nome***`.pdf`'	Composizione in PDF.
'***nome***`.ps`'	Composizione in PostScript.
'***nome***`.tex`'	Composizione transitoria in formato LaTeX.
'***nome***`.html`' '***nome***`.htm`'	Primo file della composizione in HTML.
'***nomen***`.html`' '***nomen***`.htm`'	***n***-esimo file della composizione in HTML.

File	Descrizione
'***nome_capitolo*`.html`' '***nome_capitolo*`.htm`'	Collegamento simbolico al file HTML il cui titolo corrisponde sostanzialmente al nome del collegamento stesso.
'***n*`.jpg`'	***n***-esimo file delle immagini relativo alla composizione in HTML.
'***nome_figura*`.jpg`'	Collegamento simbolico al file JPG il cui titolo corrisponde sostanzialmente al nome del collegamento stesso.
'***n*`.midi`' '***n*`.mid`'	***n***-esimo file MIDI, relativo alla composizione in HTML, generato da codice LilyPond incorporato.
'***nome_brano*`.midi`' '***nome_brano*`.mid`'	Collegamento simbolico al file MIDI il cui titolo corrisponde sostanzialmente al nome del collegamento stesso.
'***n*`.ogv`'	***n***-esimo file video Ogg relativo alla composizione in HTML.
'***nome_video*`.ogv`'	Collegamento simbolico al file OGV il cui titolo corrisponde sostanzialmente al nome del collegamento stesso.
'***n*`.ps`'	***n***-esimo file delle immagini relativo alla composizione in PostScript o PDF.
'***n*`.pdf`'	***n***-esimo file delle immagini relativo alla composizione in PostScript o PDF.
'`*~`'	File temporaneo non meglio precisato.

È bene sottolineare che il file indicato come '***nome*`.sgml`' deve essere già presente perché si possa usare Alml; inoltre, il sorgente SGML principale potrebbe a sua volta incorporare altri file SGML.

Se il sorgente SGML fa riferimento a immagini collocate in file esterni, è necessario che queste siano in uno dei formati previsti (in generale, i formati più comuni sono accettati) e che si trovino in un'altra directory rispetto a quella in cui sta il file sorgente principale.

A seconda del tipo di composizione finale, Alml converte le immagini nel formato appropriato, il più delle volte avvalendosi per questo di ImageMagick, creando una serie di file nella directory corrente. Per la composizione in PostScript e in PDF servono immagini EPS; per la composizione HTML vengono generati file in formato JPG.

I file esterni delle immagini da includere nella composizione, devono trovarsi in una directory differente da quella in cui si trova il sorgente principale, per non ritrovarli mescolati assieme a quelli che vengono generati da Alml, nella directory corrente, con nomi del tipo '***n*`.jpg`', '***n*`.ps`' o '***n*`.pdf`'.

Alle volte si possono incontrare problemi inspiegabili nell'inserimento di immagini, che si possono manifestare in modo particolare nella composizione in PDF. Spesso si superano questi problemi in modo sbrigativo usando ImageMagick e facendo un passaggio intermedio nel formato JPG, allo scopo di perdere delle informazioni. Per esempio, disponendo del file '`pippo.png`' che risulta corretto e perfettamente visibile con gli strumenti normali, ma che si comporta in modo strano nella composizione PDF, può convenire il passaggio seguente:

```
$ convert pippo.png pippo.jpg [Invio]
$ convert pippo.jpg pippo.png [Invio]
```

Al termine, il file '`pippo.jpg`' può essere eliminato.

Sintassi nell'uso del programma frontale

Il programma frontale attraverso cui si gestisce il sistema di composizione Alml è '**`alml`**':

```
alml opzioni sorgente_sgml
alml --help
alml --version
```


Come si vede dal modello sintattico, a parte i casi delle opzioni '**`--help`**' e '**`--version`**', è sempre richiesta l'indicazione di un file sorgente SGML, a cui applicare un qualche tipo di elaborazione.

Si osservi che per la composizione destinata alla stampa, è possibile lavorare **solo con i formati A4 e lettera** (8,5 in × 11 in), che possono essere orientati verticalmente oppure orizzontalmente. Eccezionalmente, per la sola composizione PostScript, è possibile selezionare il formato A5x4 verticale. Per questo, si vedano in particolare le opzioni '**`--paper`**' e '**`--paper-orientation`**'.

Tabella u64.12. Opzioni principali.

Opzione	Descrizione
`--help`	Mostra la guida rapida interna e conclude il funzionamento.
`--version`	Mostra le informazioni sulla versione e conclude il funzionamento.
`--clean`	Rimuove alcuni file temporanei abbinati al file sorgente indicato. Si tratta per la precisione di '***nome*`.pageref`', '***nome*`.diag`', '***nome*`.aux`', '***nome*`.log`', '***nome*`.sp`' e '***nome*`.sp2`'.
`--verbose`	Segnala il procedere dell'elaborazione con informazioni dettagliate. In generale tali informazioni sono ottenibili dal file '***nome*`.diag`'; tuttavia, in presenza di file sorgenti di grandi dimensioni, può servire per sapere a che punto è l'elaborazione.
`--input-encoding={latin1|utf8}`	Dichiara il formato dei file sorgenti SGML utilizzati per la composizione; in mancanza di questa opzione, il formato viene determinato in base allo stato della configurazione locale.
`--paper={a4|letter|a5x4}`	Permette di specificare le dimensioni della carta in base a un nome standard. Il formato predefinito è A4, che corrispondente alla parola chiave '**`a4`**'; il formato '**`a5x4`**' funziona solo in abbinamento a '**`--ps`**'.
`--paper-orientation=`↩ ↪`{portrait|landscape}`	Permette di specificare l'orientamento della carta.
`--static` `--dynamic`	Le due opzioni sono contrapposte. Nel primo caso si ha una composizione normale; nel secondo, se viene generato un formato PostScript o PDF, si abilitano le funzioni dinamiche per le presentazioni (in pratica, si abilita l'uso dell'elemento '**`PAUSE`**').
`--embedded-script-enable`	Abilita l'esecuzione di script incorporati nel sorgente. Trattandosi di una funzionalità che può essere pericolosa, deve essere abilitata con questa opzione, in modo esplicito.

Opzione	Descrizione
`--draft`	Quando il contesto lo permette, serve per ottenere una composizione particolare, con più informazioni utili alla correzione o alla revisione del testo. A differenza di quanto si potrebbe essere portati a pensare, in questo modo l'elaborazione è più complessa del normale, proprio per portare in risalto tali informazioni.
`--sgml-include=`***entità_parametrica***	Attraverso questa opzione, che può essere usata anche più volte, è possibile «includere» delle entità parametriche. Per la precisione, è come se nel sorgente venisse dichiarata un'entità parametrica corrispondente, assegnandole la parola chiave '**`INCLUDE`**'. Ciò viene usato per controllare l'inclusione di porzioni di sorgente, secondo le convenzioni dell'SGML.
`--page-numbering=`↩ ↪`{plain|default|tome}`	Questa opzione permette di definire in che modo gestire la numerazione delle pagine nei formati di composizione cartacei. In condizioni normali, la numerazione è realizzata attraverso sequenze differenti: una per la parte iniziale fino alla fine dell'introduzione, una per il corpo (comprese le appendici) e una finale per gli indici analitici. Assegnando la parola chiave '**`plain`**' si fa in modo che la numerazione sia unica, cosa che potrebbe essere conveniente per il formato PDF. Nel caso particolare della parola chiave '**`tome`**', si ottiene una numerazione separata dei volumi, con la conseguenza che alcuni indici, a seconda del contesto, oltre a indicare la pagina aggiungono un prefisso corrispondente al numero del volume in cui si trova.
`--sgml-syntax` `--sgml-check`	Una qualunque di queste due opzioni permette di ottenere la verifica formale del sorgente, in base al DTD.
`--sp`	Con questa opzione si vuole raggiungere solo un formato intermedio per il controllo diagnostico del funzionamento di Alml.
`--tex` `--latex`	Con questa opzione si vuole raggiungere solo un formato intermedio in LaTeX per il controllo diagnostico del funzionamento di Alml.

Opzione	Descrizione
`--dvi`	Genera un risultato in formato DVI. L'elaborazione crea una serie di file EPS e PDF per le immagini, secondo i modelli '***n***`.ps`' e '***n***`.pdf`'.
`--ps` `--postscript`	Genera un risultato in formato PostScript. L'elaborazione crea una serie di file EPS e PDF per le immagini, secondo i modelli '***n***`.ps`' e '***n***`.pdf`'; una volta ottenuto il file PostScript finale, questi file non servono più.
`--pdf`	Genera un risultato in formato PDF. L'elaborazione crea una serie di file EPS e PDF per le immagini, secondo i modelli '***n***`.ps`' e '***n***`.pdf`'; una volta ottenuto il file PDF finale, questi file non servono più.
`--html`	Genera un risultato in formato HTML, articolato in più file, dove il primo è '***nome***`.html`' e gli altri sono '***nome n***`.html`'. Inoltre, viene fatta una copia dei file delle immagini, secondo il modello '***n***`.jpg`' (le due numerazioni sono indipendenti).
`--htm`	Genera un risultato in formato HTML, simile a quello che si ottiene con '**`--html`**', dove però le estensioni dei file hanno solo tre caratteri ('`.htm`', '`.mid`', ecc.).
`--html-text`	Genera un risultato in formato HTML speciale, in un file unico, senza riferimenti a immagini esterne. Il file ottenuto può essere consultato con Links e con questo può essere convertito in un testo puro e semplice, attraverso il comando: '**`links -dump` *`nome`*`.html > ` *`nome`*`.txt`**' Oppure: '**`w3m -dump` *`nome`*`.html > ` *`nome`*`.txt`**'

Tabella u64.13. Opzioni accessorie.

Opzione	Descrizione
`--html-check` `--html401-check`	Se sono stati installati i file necessari, consente la verifica formale di un file HTML secondo le specifiche della versione 4.01.
`--html320-check`	Se sono stati installati i file necessari, consente la verifica formale di un file HTML secondo le specifiche della versione 3.2.

Opzione	Descrizione
`--xml-check`	Se sono stati installati i file necessari, consente la verifica formale di un file XML secondo le specifiche del DTD relativo (attualmente solo XHTML).

Codifica del sorgente

Il sorgente SGML usato da Alml può essere scritto secondo la codifica ISO 8859-1 (Latin-1), oppure la codifica UTF-8. In pratica, nel secondo caso si può usare la codifica universale, dove però solo una piccola porzione di punti di codifica ha una corrispondenza effettiva nella composizione.

Allo stato attuale è possibile scrivere usando lingue che si avvalgono dell'alfabeto latino, il greco e il russo, come si può vedere meglio nel capitolo u79.

Esiste comunque la necessità che tutti i file che compongono il sorgente SGML siano scritti nella stessa codifica: tutti ISO 8859-1, oppure tutti UTF-8. In generale, non si presenta la necessità di usare la codifica UTF-8, nemmeno quando si volesse selezionare un carattere a cui non risulta associata alcuna entità standard. Infatti, in questi casi, si può usare un riferimento numerico nella forma:

```
&#xhhhh;
```

In pratica, volendo fare riferimento al punto di codifica U+266E in forma numerica (♮), si potrebbe scrivere '**`♮`**'.

Dal momento che non c'è un modo pratico per distinguere automaticamente se un file sia scritto usando l'una o l'altra codifica, è possibile usare l'opzione '**`--input-encoding`**' per specificarlo espressamente. Tuttavia, se questa opzione non viene usata, Alml fa delle congetture basandosi sullo stato attuale della variabile di ambiente ***LANG*** e delle variabili ***LC_**; in pratica, tenta di determinarlo dalla configurazione locale.

Organizzare un file-make o uno script

Un file-make personalizzato può facilitare l'uso di Alml. Viene proposto un esempio elementare, riferito al file '`example.sgml`', in cui si può vedere anche l'utilizzo proposto di '**`alml`**'.

```
# file name prefix.
DOC_PREFIX=example

# Notice that "text" generates an HTML file with the same name
# for the first HTML page. This is why it is before the standard
# HTML typesetting.
#
all:    \
clean   \
text    \
html    \
ps      \
pdf

clean:
        @echo "Cleaning..."                          ; \
        find . -name core    -exec rm -f \{\} \;     ; \
        rm -f $(DOC_PREFIX)*.tex                     ; \
        rm -f $(DOC_PREFIX)*.dvi                     ; \
        rm -f $(DOC_PREFIX)*.sp                      ; \
        rm -f $(DOC_PREFIX)*.sp2                     ; \
        rm -f $(DOC_PREFIX)*.ps                      ; \
        rm -f $(DOC_PREFIX)*.pdf                     ; \
        rm -f $(DOC_PREFIX)*.txt                     ; \
        rm -f $(DOC_PREFIX)*.log                     ; \
        rm -f $(DOC_PREFIX)*.aux                     ; \
        rm -f $(DOC_PREFIX)*.tmp                     ; \
        rm -f $(DOC_PREFIX)*.diag                    ; \
        rm -f $(DOC_PREFIX)*.pageref                 ; \
        rm -f $(DOC_PREFIX)*.pageloc                 ; \
        rm -f *.html *.htm                           ; \
        rm -f *.bak                                  ; \
        rm -f *.jpg                                  ; \
        rm -f *.ps                                   ; \
        rm -f *.midi *.mid                           ; \
        rm -f *\~

check:
        @alml --sgml-check              \
              --verbose                 \
              $(DOC_PREFIX).sgml

dvi:
        @alml --dvi                     \
              --verbose                 \
              $(DOC_PREFIX).sgml

ps:
        @alml --ps                      \
              --verbose                 \
              $(DOC_PREFIX).sgml

pdf:
        @alml --pdf                     \
              --verbose                 \
              --page-numbering=plain    \
              $(DOC_PREFIX).sgml

html:
        @alml --html                    \
              --verbose                 \
              $(DOC_PREFIX).sgml

htm:
        @alml --htm                     \
              --verbose                 \
              $(DOC_PREFIX).sgml

text:
        @alml --html-text               \
              --verbose                 \
              $(DOC_PREFIX).sgml        ; \
        w3m -dump                       \
            $(DOC_PREFIX).html          \
            > $(DOC_PREFIX).txt
```

Si può osservare in particolare l'obiettivo '**`clean`**' che elimina tutti i file non indispensabili e in particolare tutti i file il cui nome termina per '`.html`' e per '`.ps`'.

Se per esempio si utilizza il comando '**`make ps`**', si ottiene la composizione in PostScript, generando in particolare il file '`example.ps`'.

Uno script da usare sostanzialmente come il file-make proposto, potrebbe essere realizzato così:

```
#!/bin/sh
#
ACTION=$1
#
DOC_PREFIX=example
#
if   [ "$ACTION" = "" ]
then
    echo "Please, specify an action:"
    echo "$0 ACTION"
    exit
elif [ "$ACTION" = "clean" ]
then
    echo "Cleaning..."
    find . -name core    -exec rm -f \{\} \;
    rm -f $DOC_PREFIX*.tex
    rm -f $DOC_PREFIX*.dvi
    rm -f $DOC_PREFIX*.sp
    rm -f $DOC_PREFIX*.sp2
    rm -f $DOC_PREFIX*.ps
    rm -f $DOC_PREFIX*.pdf
    rm -f $DOC_PREFIX*.txt
    rm -f $DOC_PREFIX*.log
    rm -f $DOC_PREFIX*.aux
    rm -f $DOC_PREFIX*.tmp
    rm -f $DOC_PREFIX*.diag
    rm -f $DOC_PREFIX*.pageref
    rm -f $DOC_PREFIX*.pageloc
    rm -f *.html *.htm
    rm -f *.bak
    rm -f *.jpg
    rm -f *.ps
    rm -f *.midi *.mid
    rm -f *\~
    #
elif [ "$ACTION" = "check" ]
then
    alml --sgml-check               \
         --verbose                  \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "dvi" ]
then
    alml --dvi                      \
         --verbose                  \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "ps" ]
then
    alml --ps                       \
         --verbose                  \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "pdf" ]
```

```
then
    alml --pdf                      \
         --verbose                  \
         --page-numbering=plain     \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "html" ]
then
    alml --html                     \
         --verbose                  \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "htm" ]
then
    alml --htm                      \
         --verbose                  \
         $DOC_PREFIX.sgml
    #
elif [ "$ACTION" = "text" ]
then
    alml --html-text                \
         --verbose                  \
         $DOC_PREFIX.sgml
    w3m -dump                       \
        $DOC_PREFIX.html            \
        > $DOC_PREFIX.txt
```

Formati particolari

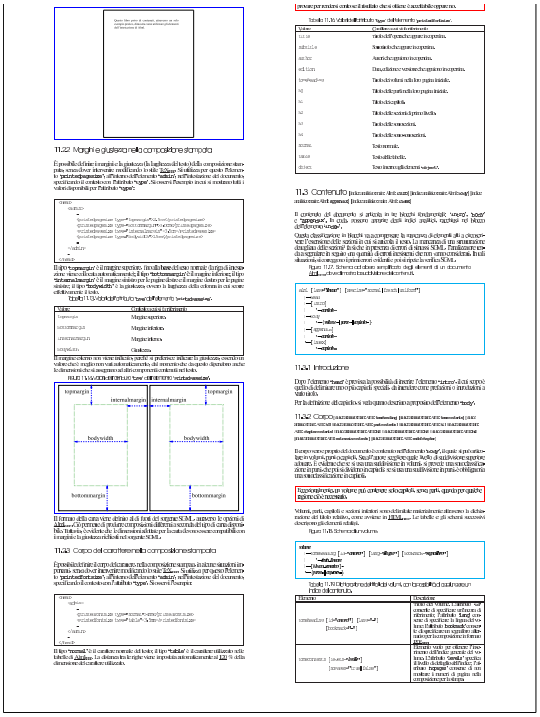
L'opzione '**`--paper`**' di '**`alml`**' consente di definire il formato della pagina per la composizione destinata alla stampa. Generalmente si possono usare solo i formati A4 e lettera, rispettivamente con le opzioni '**`--paper=a4`**' e '**`--paper=letter`**'. Eccezionalmente, quando si intende generare un formato PostScript, è possibile produrre un formato A5x4 verticale, ovvero 21 cm × 59,4 cm ('**`--paper=a5x4`**').

Il formato A5x4 può essere utile, rielaborando il file PostScript in modo da ridurlo e da accoppiarlo su un foglio A4 singolo. Per ottenere questo risultato ci si può avvalere di '**`alml-extra`**', usandolo come nel comando seguente:

```
$ alml-extra --a5x4-to-a7x4-2-a4 nome.ps [Invio]
```

In tal caso, il file '***nome***`.ps`' è il file PostScript in formato A5x4 e si ottiene il file '***nome***`.a7x4-2-a4.ps`', in formato A4, che in pratica contiene due colonne formato A7x4 (10,5 cm × 29,7 cm).

Figura u64.16. Esempio di come può apparire una pagina che contiene due colonne in formato A7x4.



Progetti di documentazione che utilizzano il formato di Alml

L'elenco successivo riporta alcuni progetti di documentazioni che utilizzano Alml:

- Gianluca Giusti, *Programmare in PHP*
  http://www.urcanet.it/brdp/php_manual/
- Gaetano Paolone, *Linux domande e risposte*
  http://linuxfaq.it
- Fulvio Ferroni, *Programmazione dei socket di rete in GNU/Linux*
  http://linuxdidattica.org/docs/altre_scuole/planck/socket/
- Fulvio Ferroni, *Samba e OpenLDAP*
  http://linuxdidattica.org/docs/altre_scuole/planck/samba/
- Massimo Piai, *Informatica per sopravvivere*
  http://linuxdidattica.org/piai/xs/

[1] Nelle distribuzioni Debian si tratta del pacchetto '**`liblocale-gettext-perl`**'.

Il documento secondo Alml

Organizzazione generale 411
Dalla copertina all'indice generale 412
Esempio quasi completo per la compilazione dell'intestazione 415
Margini e giustezza nella composizione stampata 417
Corpo del carattere nella composizione stampata 418
Contenuto 419
Introduzione 419
Corpo 419
Appendici 422
Indici analitici 422
Suddivisioni speciali 423
Documento multilingua 425
Cambiamento temporaneo del linguaggio 426
Definizione alternativa della suddivisione del documento .. 426

Il DTD di Alml è organizzato per gestire documenti molto grandi, che possono essere suddivisi in volumi, parti e capitoli. Tuttavia, la suddivisione in volumi o in parti resta facoltativa, mentre la divisione in capitoli è obbligatoria.

Quando devono essere indicate delle dimensioni che prevedono la specificazione dell'unità di misura, si usano le sigle elencate nella tabella u65.1.

Tabella u65.1. Sigle delle unità di misura utilizzabili con Alml.

Sigla	Unità di misura corrispondente
`pt`	Punti tipografici corrispondenti a 1/72,27 di pollice.
`bp`	Punti tipografici corrispondenti a 1/72 di pollice (*big point*).
`pc`	Pica corrispondenti a 1/6 di pollice.
`in`	Pollici.
`cm`	Centimetri.
`mm`	Millimetri.

Organizzazione generale

Secondo il DTD di Alml, il documento ha una struttura generale ben definita:

```
<!DOCTYPE ALML PUBLIC "-//D.G.//DTD Alml//EN">
<alml>
<head>
…
</head>
[<intro>
…
</intro>]
<body>
…
</body>
[<appendix>
…
</appendix>]
[<index>
…
</index>]
</alml>
```

In questa struttura, gli elementi '**head**' e '**body**' sono obbligatori, mentre gli altri possono essere omessi, se non sono necessari.

Si può intuire il senso della cosa: l'elemento '**head**' serve a contenere informazioni amministrative, oltre a ciò che deve apparire nelle primissime pagine (il titolo dell'opera, il copyright ecc.); l'elemento '**intro**' permette di inserire dei capitoli speciali da trattare come introduzioni o prefazioni, che come tali non risultano numerate; l'elemento '**body**' permette di inserire capitoli, oppure parti, o volumi; l'elemento '**appendix**' permette di inserire capitoli da trattare come appendici, numerate convenzionalmente in modo letterale; infine, l'elemento '**index**' permette di inserire capitoli speciali per l'inclusione degli indici analitici.

Figura u65.2. Schema ad albero degli elementi principali di un documento Alml.

```
alml [lang="lingua"] [spacing="normal|french|uniform"]
  |--head
  |--[intro]
  |--body
  |--[appendix]
  '--[index]
```

Dalla copertina all'indice generale

L'elemento che delimita il documento nella sua interezza, '**alml**', può contenere due attributi facoltativi: '**lang**' e '**spacing**'. L'attributo '**lang**' permette di definire il linguaggio generale con cui è stato scritto il documento, attraverso una sigla secondo lo standard ISO 639 (tabella 13.4), ma se le informazioni su un certo linguaggio non sono disponibili, si applicano comunque le convenzioni inglesi.

L'attributo '**spacing**' permette di definire il modo in cui vengono gestiti gli spazi alla fine dei periodi (dopo il punto fermo). Assegnando la parola chiave '**normal**', si ottiene la spaziatura normale della convenzione inglese, in cui lo spazio dopo un punto ha una larghezza maggiore degli altri; in alternativa, assegnando la parola chiave '**uniform**', oppure '**french**', si ottiene una spaziatura uniforme, come richiede la tradizione tipografica italiana e anche di altri paesi.

In generale, un documento scritto in lingua italiana dovrebbe utilizzare l'elemento '**alml**' in questo modo:

```
<alml lang="it" spacing="uniform">
```

La figura u65.4 e la tabella u65.5 mostrano in breve l'elenco degli elementi che riguardano l'intestazione del documento; cosa che contiene tutte le informazioni per realizzare la copertina, fino ad arrivare all'indice generale.

Figura u65.4. Schema ad albero degli elementi di un documento Alml, con il dettaglio dell'intestazione.

```
alml [lang="lingua"] [spacing="normal|french|uniform"]
  |--head
  |    |--[admin]
  |    |    |--[description]
  |    |    |--[keywords]
  |    |    |--[htmlmeta name="nome" lang="linguaggio"]…
  |    |    |--[printedfontsize type="contesto"]…
  |    |    |--[printedpagesize type="contesto"]…
  |    |    |--[chapterdefinition]
  |    |    |--[partdefinition]
  |    |    '--[tomedefinition]
  |    |--title
  |    |--[shorttitle]
  |    |--[subtitle]…
  |    |--author…
  |    |--date
  |    |--[edition]
  |    |--[version]
  |    |--[frontcovertop]
  |    |--[abstract]
  |    |--[frontcoverbottom]
  |    |--[backcover]
  |    |--[textbeforelegal]
  |    |--legal
  |    |--[dedications]
  |    |--[textafterdedications]
  |    '--[maincontents levels="n" nopages="true|false"]
  |--[intro]
  |--body
  |--[appendix]
  '--[index]
```

Tabella u65.5. Elementi SGML dalla copertina all'indice generale.

Elemento	Descrizione
`alml [lang="…"] [spacing="…"]`	Contenitore del documento. L'attributo '**lang**' può contenere la sigla del linguaggio espressa secondo lo standard ISO 639. L'attributo '**spacing**' può contenere una parola chiave, a scelta tra: '**normal**', '**french**' e '**uniform**'.
`head`	Intestazione del documento.
`admin`	Informazioni amministrative.
`description`	Descrizione in breve del documento.
`keywords`	Elenco di parole chiave.
`htmlmeta name="…" lang="…"`	Contenuto di un elemento HTML '**META**'. Gli attributi '**name**' e '**lang**' vanno usati nello stesso modo previsto per l'elemento '**META**' di HTML.
`chapterdefinition`	Definizione alternativa del capitolo.
`partdefinition`	Definizione alternativa della parte.
`tomedefinition`	Definizione alternativa del volume.
`printedfontsize type="…"`	Corpo del carattere in punti. Il carattere a cui si fa riferimento è quello indicato nell'attributo '**type**'.
`printedpagesize type="…"`	Dimensione di quanto indicato nell'attributo '**type**', che in generale si riferisce alla definizione dei margini e della giustezza.
`title`	Titolo del documento.
`shorttitle`	Sigla o abbreviazione del titolo dell'opera; è utile nella composizione HTML.
`subtitle`	Sottotitolo.
`author`	Autore.
`date`	Data del lavoro.
`edition`	Edizione, da usare se questa è diversa dalla data.
`version`	Versione, se la si vuole indicare in modo diverso dalla data di edizione.
`frontcovertop`	Blocco che precede il titolo.

Elemento	Descrizione
`abstract`	Descrizione del contenuto. **Si osservi che attualmente questa informazione non viene utilizzata in fase di composizione.**
`frontcoverbottom`	Testo aggiuntivo di copertina, da mostrare dopo il titolo e dopo le altre indicazioni standard.
`backcover`	Contenuto della copertina finale.
`textbeforelegal`	Testo prima delle informazioni legali.
`legal`	Informazioni legali (copyright, condizioni, ecc.).
`dedications`	Pagina delle dediche.
`textafterdedications`	Testo successivo alle dediche.
`maincontents` [`levels="…"`] [`nopages="…"`]	Inserimento dell'indice generale, specificando il titolo da dare a tale indice. L'attributo '**`levels`**' specifica il livello di dettaglio dell'indice. L'attributo '**`nopages`**' specifica se si vogliano vedere i numeri di pagina come riferimento nella composizione stampata; può assumere i valori '**`true`**' o '**`false`**'.

Si può osservare che tutte le informazioni sono contenute nell'elemento '**`head`**', all'inizio del quale prende posto un altro «contenitore» denominato '**`admin`**'. Al suo interno sono previsti elementi relativi a informazioni amministrative, in particolare '**`description`**' e '**`keywords`**', il cui scopo è quello di generare degli elementi '**`META`**' corrispondenti nella composizione HTML:

```
<HEAD>
    …
    <META NAME="Description" CONTENT="An example for Alml documentation system">
    <META NAME="Keywords" CONTENT="SGML, XML, HTML, Alml">
    …
</HEAD>
```

Inoltre, si possono aggiungere anche altri elementi '**`META`**' di HTML, attraverso l'elemento '**`HTMLMETA`**', come si vede nell'esempio seguente:

```
<head>
    <admin>
        <description>GNU/Linux e altro software libero</description>

        <keywords>Linux, GNU/Linux, Unix, software, software libero,
        free software</keywords>

        <htmlmeta name="Resource-type" lang="en">Document</htmlmeta>
        <htmlmeta name="Revisit-after" lang="en">15 days</htmlmeta>
        <htmlmeta name="Robots">ALL</htmlmeta>
    </admin>
    …
    …
</head>
```

Gli elementi '**`chapterdefinition`**', '**`partdefinition`**' e '**`tomedefinition`**' vengono descritti più avanti in questo capitolo (sezione u0.5).

L'elemento '**`printedfontsize`**' consente di definire l'altezza del carattere indicato attraverso l'attributo '**`type`**', per la composizione stampata.

L'elemento '**`printedpagesize`**' consente di definire I margini e la giustezza per la composizione stampata, in base al contesto indicato dall'attributo '**`type`**'.

L'elemento '**`title`**' serve a indicare il titolo del documento; gli elementi eventuali '**`subtitle`**' permettono di inserire dei sottotitoli successivi.

L'elemento '**`abstract`**', facoltativo, permette l'inserimento di una descrizione, più o meno articolata, composta da blocchi di testo. Tuttavia, questa informazione non viene usata in fase di composizione.

Successivamente è possibile inserire uno o più elementi '**`author`**', uno per il nominativo di ogni coautore.

Gli elementi '**`date`**', '**`edition`**' e '**`version`**', servono per indicare una data, un'edizione e una versione del lavoro. In generale è sufficiente l'uso dell'elemento '**`data`**'.

L'elemento '**`frontcovertop`**' permette l'inserzione di blocchi prima del titolo; così, l'elemento '**`frontcoverbottom`**' consente di fare la stessa cosa dopo il titolo e le altre indicazioni standard. L'elemento '**`backcover`**' permette di definire il contenuto della copertina finale.

Gli elementi successivi riguardano la seconda pagina assoluta e quelle successive.

Nella seconda pagina appaiono di solito le informazioni sul copyright, nella parte bassa, mentre nella parte superiore potrebbero esserci altre informazioni, come una breve descrizione degli autori. L'elemento '**`textbeforelegal`**' permette di inserire blocchi di testo da collocare nella prima parte della seconda pagina, mentre l'elemento '**`legal`**' è fatto per le informazioni legali, a partire dal copyright.

Dopo le informazioni «legali» è possibile inserire una pagina di dediche, attraverso l'elemento '**`dedications`**'. Eventualmente, se necessario, è possibile aggiungere altre notizie all'interno dell'elemento '**`textafterdedications`**' che segue le dediche.

Infine, è possibile collocare l'elemento '**`maincontents`**' per ottenere l'inserimento dell'indice generale. L'attributo '**`levels`**' permette di definire il livello di dettaglio desiderato dell'indice: il numero zero rappresenta il minimo e fa in modo di ottenere informazioni fino alle parti, mentre valori superiori aumentano il dettaglio. Assegnando all'attributo '**`nopages`**' il valore '**`true`**', si richiedere espressamente l'eliminazione dei riferimenti ai numeri di pagina; cosa che può essere utile soltanto nella composizione per la stampa. All'interno dell'elemento si inserisce il titolo da dare all'indice.

Esempio quasi completo per la compilazione dell'intestazione

Viene mostrato qui un esempio quasi completo dell'uso degli elementi che si inseriscono all'interno di '**`head`**' (il file in questione dovrebbe essere disponibile presso *allegati/a2/alml-esempio-intestazione.sgml*). Di proposito, il contenuto del documento è completamente mancante, nel senso che l'elemento '**`body`**' è vuoto.

```
<!DOCTYPE ALML PUBLIC "-//D.G.//DTD Alml//EN">
<alml lang="it" spacing="uniform">
<head>
    <admin>
        <description>Compilazione di un'intestazione con Alml</description>
        <keywords>Alml, SGML, composizione</keywords>
        <printedfontsize type="title">20mm</printedfontsize>
        <printedfontsize type="subtitle">8mm</printedfontsize>
        <printedfontsize type="author">8mm</printedfontsize>
        <printedfontsize type="edition">8mm</printedfontsize>
        <printedfontsize type="normal">7mm</printedfontsize>
    </admin>
    <title>Intestazione</title>
    <shorttitle>int</shorttitle>
    <subtitle>Come iniziare con Alml</subtitle>
    <author>Pinco Pallino</author>
    <date>1111.11.11</date>
    <edition>1212.12.12</edition>
    <version>1.1</version>
    <frontcovertop>
        <p>i libri di Alml</p>
    </frontcovertop>
    <abstract>
        <p>La compilazione delle informazioni di un documento
        è sempre complicato all'inizio dello studio di un sistema
        SGML o XML di composizione.</p>
        <p>Questo libro, attraverso un esempio pratico, spiega come
        utilizzare proprio gli elementi dell'intestazione.</p>
    </abstract>
    <frontcoverbottom>
        <p>!"£$%&amp;amp;/()=?^*+°§</p>
```

```
        <p>!"£$%&amp;/()=?^*+°§</p>
        <p>!"£$%&amp;/()=?^*+°§</p>
        <p>!"£$%&amp;/()=?^*+°§</p>
    </frontcoverbottom>
    <backcover>
        <p>Questo libro privo di contenuti, attraverso un solo esempio
        pratico, dimostra come utilizzare gli elementi dell'intestazione
        di Alml.</p>
    </backcover>
    <textbeforelegal>
        <p>Pinco Pallino è laureato in scienza del vuoto mentale
        e insegna nullafacenza applicata.</p>
    </textbeforelegal>
    <legal>
        <p>Copyright &copy; Pinco Pallino, &lt;pinco.pallino@brot.dg&gt;</p>
        <p>Permission is granted to copy, distribute and/or modify this
        document under the terms of the GNU Free Documentation License,
        Version 1.1 or any later version published by the Free Software
        Foundation; with no Invariant Sections, with no Front-Cover
        Texts, and with no Back-Cover Texts. A copy of the license is
        included in the section entitled "GNU Free Documentation
        License".</p>
    </legal>
    <dedications>
        <p>Alla mia bella Gigia, con tanto amore.</p>
    </dedications>
    <textafterdedications>
        <p>Sette, sei, cinque, quattro, tre, due, uno,... via!</p>
    </textafterdedications>
    <maincontents levels="2">Indice generale</maincontents>
</head>
<body>
</body>
</alml>
```

Nelle figure successive viene mostrato il risultato della composizione in un formato PostScript o PDF, in modo indifferente. Si suppone che il file sorgente sia stato chiamato '`head.sgml`' e che sia stato usato uno dei due comandi seguenti:

```
$ alml --ps alml-esempio-intestazione.sgml [Invio]
```

```
$ alml --pdf alml-esempio-intestazione.sgml [Invio]
```

Il risultato che si dovrebbe ottenere, in formato PDF, può essere prelevato presso *allegati/a2/alml-esempio-intestazione.pdf* (viene distribuito assieme all'edizione HTML dell'opera).

Figura u65.9. La copertina e la pagina del colofone (che appare subito dopo la copertina). Nella pagina della copertina si può osservare che: in alto, prima del titolo, viene messo il contenuto di '**`frontcovertop`**'; disponendo della versione dell'edizione, appare il contenuto degli elementi '**`version`**' e '**`edition`**'; nella parte sottostante appare il contenuto dell'elemento '**`frontcoverbottom`**'. Nella pagina del colofone si vede in alto il contenuto di '**`textbeforelegal`**' e in basso il contenuto di '**`legal`**'.

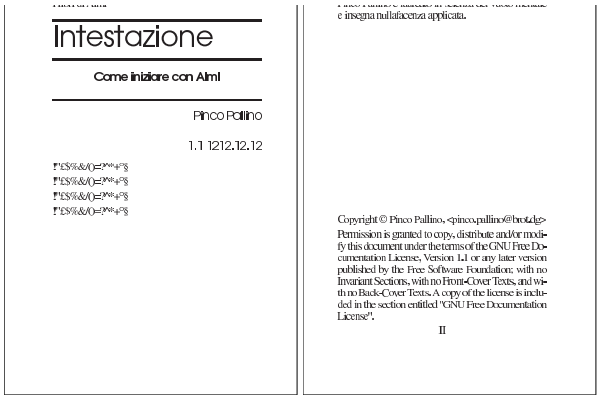


Figura u65.10. La pagina delle dediche, ovvero la pagina associata all'elemento '**`dedications`**', assieme alla pagina successiva, corrispondente al contenuto dell'elemento '**`textafterdedications`**'.

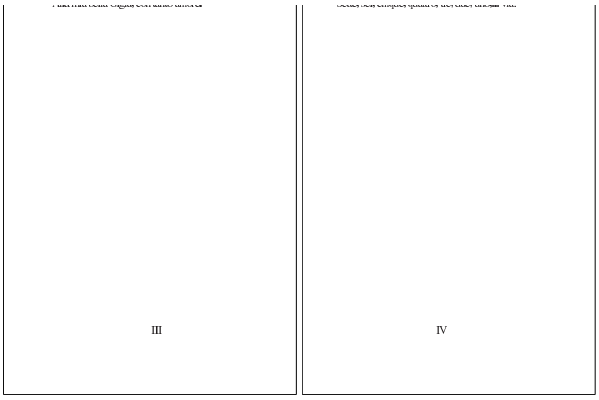
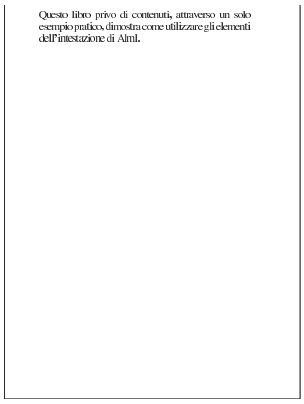


Figura u65.11. La quarta di copertina (copertina posteriore), corrispondente all'elemento '**`backcover`**'.



Margini e giustezza nella composizione stampata

È possibile definire i margini e la giustezza (la larghezza del testo) della composizione stampata, senza dover intervenire modificando lo stile TeX. Si utilizza per questo l'elemento '**`printedpagesize`**', all'interno dell'elemento '**`admin`**', nell'intestazione del documento, specificando il contesto con l'attributo '**`type`**'. Si osservi l'esempio in cui si mostrano tutti i valori disponibili per l'attributo '**`type`**':

```
<head>
    <admin>
        ...
        <printedpagesize type="topmargin">2.5cm</printedpagesize>
        <printedpagesize type="bottommargin">3.0cm</printedpagesize>
        <printedpagesize type="internalmargin">3.5cm</printedpagesize>
        <printedpagesize type="bodywidth">15cm</printedpagesize>
        ...
    </admin>
    ...
</head>
```

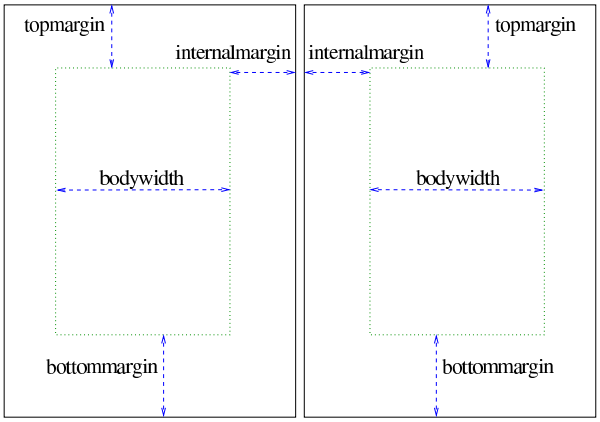
Il tipo '**`topmargin`**' è il margine superiore, fino alla **base** del testo normale (la riga di intestazione viene collocata automaticamente); il tipo '**`bottommargin`**' è il margine inferiore; il tipo '**`internalmargin`**' è il margine sinistro per le pagine destre e il margine destro per le pagine sinistre; il tipo '**`bodywidth`**' è la giustezza, ovvero la larghezza della colonna in cui scorre effettivamente il testo.

Tabella u65.13. Valori dell'attributo '**type**' dell'elemento '**printedpagesize**'.

Valore	Contesto a cui si fa riferimento
topmargin	Margine superiore.
bottommargin	Margine inferiore.
internalmargin	Margine interno.
bodywidth	Giustezza.

Il margine esterno non viene indicato, perché si preferisce indicare la giustezza, essendo un valore che è meglio non vari automaticamente, dal momento che da questo dipendono anche le dimensioni che si assegnano ad altri componenti contenuti nel testo.

Figura u65.14. Valori dell'attributo '**type**' dell'elemento '**printedpagesize**'.



Il formato della carta viene definito al di fuori del sorgente SGML, attraverso le opzioni di Alml. Ciò permette di produrre composizioni differenti a seconda del tipo di carta disponibile. Tuttavia, è evidente che le dimensioni adottate per la carta devono essere compatibili con i margini e la giustezza richiesti nel sorgente SGML.

Corpo del carattere nella composizione stampata

È possibile definire il corpo del carattere, nella composizione stampata, in alcune situazioni importanti, senza dover intervenire modificando lo stile TeX. Si utilizza per questo l'elemento '**printedfontsize**', all'interno dell'elemento '**admin**', nell'intestazione del documento, specificando il contesto con l'attributo '**type**'. Si osservi l'esempio:

```
<head>
    <admin>
        …
        <printedfontsize type="normal">4mm</printedfontsize>
        <printedfontsize type="table">3.5mm</printedfontsize>
        …
    </admin>
    …
</head>
```

Il tipo '**normal**' è il carattere normale del testo; il tipo '**table**' è il carattere utilizzato nelle tabelle di Alml. La distanza tra le righe viene impostata automaticamente al 120 % della dimensione del carattere utilizzato.

La dimensione del carattere deve essere armoniosa rispetto al resto del documento. Bisogna provare per rendersi conto se il risultato che si ottiene è accettabile oppure no.

Tabella u65.16. Valori dell'attributo '**type**' dell'elemento '**printedfontsize**'.

Valore	Carattere a cui si fa riferimento
title	Titolo dell'opera che appare in copertina.
subtitle	Sottotitolo che appare in copertina.
author	Autori che appaiono in copertina.
edition	Data, edizione e versione che appaiono in copertina.
tomeheading	Titolo dei volumi nella loro pagina iniziale.
h0	Titolo delle parti nella loro pagina iniziale.
h1	Titolo dei capitoli.
h2	Titolo delle sezioni di primo livello.
h3	Titolo delle sottosezioni.
h4	Titolo delle sotto-sottosezioni.
normal	Testo normale.
table	Testo delle tabelle.
object	Testo interno agli elementi '**object**'.

Contenuto

Il contenuto del documento si articola in tre blocchi fondamentali: '**intro**', '**body**' e '**appendix**'. In coda, possono apparire degli indici analitici, racchiusi nel blocco dell'elemento '**index**'.

Questa classificazione in blocchi va a compensare la mancanza di elementi atti a circoscrivere l'estensione delle sezioni in cui si articola il testo. La mancanza di una strutturazione dettagliata delle sezioni[1] fa sì che in presenza di errori di sintassi SGML, l'analizzatore tenda a segnalare in seguito una quantità di errori inesistenti che non vanno considerati. In tali situazioni, si correggono i primi errori evidenti e poi si ripete la verifica SGML.

Figura u65.17. Schema ad albero semplificato degli elementi di un documento Alml, dove di mostra la suddivisione dei contenuti.

```
alml [lang="lingua"] [spacing="normal|french|uniform"]
  |--head
  |--[intro]
  |     '--capitolo…
  |--body
  |     '--{volume… | parte… | capitolo…}
  |--[appendix]
  |     '--capitolo…
  '--[index]
        '--capitolo…
```

Introduzione

Dopo l'elemento '**head**' è prevista la possibilità di inserire l'elemento '**intro**', il cui scopo è quello di delimitare uno o più capitoli speciali, da intendere come prefazioni o introduzioni a vario titolo.

Per la definizione del capitolo, si veda quanto descritto a proposito dell'elemento '**body**'.

Corpo

Il corpo vero e proprio del documento è contenuto nell'elemento '**body**', il quale si può articolare in volumi, parti o capitoli. Sta all'autore scegliere quale livello di suddivisione superiore adottare. È evidente che se si usa una suddivisione in volumi, si prevede una sottoclassificazione in parti, che poi si dividono in capitoli; se si usa una suddivisione in parti, è obbligatoria una sottoclassificazione in capitoli.

Eccezionalmente, un volume può contenere solo capitoli, senza parti, quando per qualche ragione ciò è necessario.

Volumi, parti, capitoli e sezioni inferiori sono delimitate materialmente attraverso la dichiarazione del titolo relativo, come avviene in HTML. Le tabelle e gli schemi successivi descrivono gli elementi relativi.

Figura u65.18. Schema di un volume.

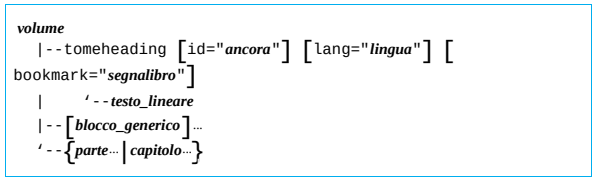


Tabella u65.19. Dichiarazione dei titoli dei volumi, con la possibilità di aggiungere un indice del contenuto.

Elemento	Descrizione
`tomeheading [id="ancora"] [lang="…"] [bookmark="…"]`	Titolo del volume. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**lang**' consente di specificare la lingua del volume; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`tomecontents [levels="livelli"] [nopages="true|false"]`	Elemento vuoto per ottenere l'inserimento dell'indice generale del volume. L'attributo '**levels**' specifica il livello di dettaglio dell'indice; l'attributo '**nopages**' consente di non mostrare i numeri di pagina nella composizione per la stampa.

Figura u65.20. Schema di una parte.

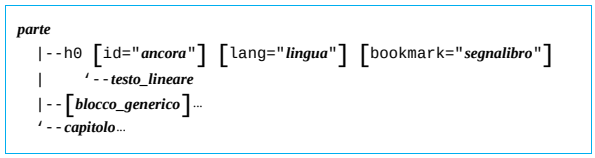


Tabella u65.21. Dichiarazione dei titoli delle parti, con la possibilità di aggiungere un indice del contenuto.

Elemento	Descrizione
`h0 [id="ancora"] [lang="…"] [bookmark="…"]`	Titolo della parte. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**lang**' consente di specificare la lingua della parte; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`partcontents [levels="livelli"] [nopages="true|false"]`	Elemento vuoto per ottenere l'inserimento dell'indice generale della parte. L'attributo '**levels**' specifica il livello di dettaglio dell'indice; l'attributo '**nopages**' consente di non mostrare i numeri di pagina nella composizione per la stampa.

Figura u65.22. Schema di un capitolo e della sua suddivisione inferiore.

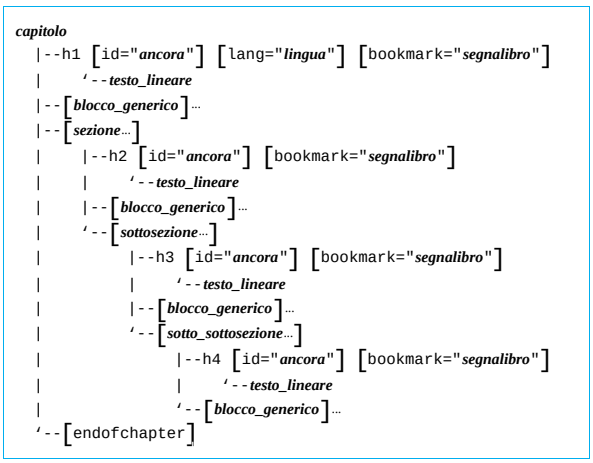


Tabella u65.23. Elementi relativi alla definizione di un capitolo.

Elemento	Descrizione
`h1 [id="ancora"] [lang="…"] [bookmark="…"]`	Titolo del capitolo. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**lang**' consente di specificare la lingua del capitolo; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`chaptercontents [levels="livelli"] [nopages="true|false"]`	Elemento vuoto per ottenere l'inserimento dell'indice generale del capitolo. L'attributo '**levels**' specifica il livello di dettaglio dell'indice; l'attributo '**nopages**' consente di non mostrare i numeri di pagina nella composizione per la stampa.
`h2 [id="ancora"] [bookmark="…"]`	Titolo della sezione. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.

Elemento	Descrizione
`h3 [id="`*ancora*`"] [bookmark="…"]`	Titolo della sottosezione. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`h4 [id="`*ancora*`"] [bookmark="…"]`	Titolo della sotto-sottosezione. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`extramaincontents [levels="`*livelli*`"] [nopages="true|false"]`	Elemento vuoto per ottenere l'inserimento di un indice generale complessivo. L'attributo '**levels**' specifica il livello di dettaglio dell'indice; l'attributo '**nopages**' consente di non mostrare i numeri di pagina nella composizione per la stampa.
`endofchapter`	Testo lineare da inserire, eventualmente, alla fine di un capitolo, con delle note particolari.

Nella parte iniziale delle classificazioni principali (volumi, parti e capitoli), è possibile collocare la richiesta di inserimento di un indice generale specifico. Si ottiene questo con gli elementi: '**tomecontents**', '**partcontents**' e '**chaptercontents**' (è disponibile anche l'elemento '**extramaincontents**' che riguarda l'opera intera e può essere collocato ovunque). Ognuno di questi elementi prevede l'attributo '**levels**', con il quale è possibile stabilire il livello di dettaglio di tali indici, tenendo presente che con il numero zero si ottengono voci fino alle parti, con uno si ottengono anche i capitoli, mentre con valori superiori si accede alle sezioni di livello inferiore. Anche in questo caso è possibile inibire la segnalazione delle pagine (nel caso di composizione per la stampa), utilizzando l'attributo '**nopages**'.

L'elemento '**endofchapter**' avrebbe lo scopo di consentire l'inserimento di una riga di informazioni alla fine del capitolo; precisamente, nella composizione per la stampa, alla base dell'ultima pagina del capitolo. Purtroppo, però, in presenza di riquadri fluttuanti può succedere di vedere il contenuto dell'elemento '**endofchapter**' alla fine di una pagina, mentre nelle successive vengono collocati i riquadri fluttuanti rimasti in sospeso; inoltre, può capitare di avere una pagina completamente vuota, ma contenente soltanto quanto inserito nell'elemento '**endofchapter**'.

Appendici

Dopo il corpo è possibile inserire l'elemento '**appendix**', il cui scopo è quello di delimitare uno o più capitoli speciali, da intendere come appendici.

Indici analitici

Alml consente la definizione di diversi tipi di indici analitici. Per questi è previsto uno spazio speciale collocato dopo le appendici, se ci sono, o in caso contrario subito dopo il corpo. Si tratta dell'elemento '**index**', che prevede l'inserimento di capitoli, come nel caso delle appendici.

L'inserimento di un elenco riferito a un indice analitico particolare si ottiene con l'elemento vuoto '**printindex**'. Viene descritto meglio in seguito l'uso di questo elemento, perché Alml è in grado di gestire più indici analitici differenti.

Suddivisioni speciali

Oltre alle suddivisioni standard nella forma '**h***n*', ne sono disponibili altre per scopi particolari. Sono previsti capitoli speciali per le presentazioni (diapositive o lucidi per lavagna luminosa), i prospetti schematici riassuntivi (tavole sintetiche e simili), i questionari (per le verifiche didattiche), oltre a due tipi di sezioni per domande e risposte.

Figura u65.24. Schema di un capitolo speciale per diapositive.

```
capitolo
  |--slideh1 [id="ancora"] [lang="lingua"] [
bookmark="segnalibro"]
  |    '--testo_lineare
  |--[blocco_generico]…
  '--[endofchapter]
```

Figura u65.25. Schema di un capitolo speciale per schede informative generiche.

```
capitolo
  |--sheeth1 [id="ancora"] [lang="lingua"] [
bookmark="segnalibro"]
  |    '--testo_lineare
  |--[blocco_generico]…
  '--[endofchapter]
```

Figura u65.26. Schema di un capitolo contenente domande e risposte.

```
capitolo
  |--h1 [id="ancora"] [lang="lingua"] [bookmark="segnalibro"]
  |    '--testo_lineare
  |--[blocco_generico]…
  |--[sezione…]
  |    |--faqh2 [id="ancora"] [bookmark="segnalibro"]
  |    |    '--testo_lineare
  |    |--qh2 [id="ancora"] [bookmark="segnalibro"]
  |    |    '--testo_lineare
  |    |--[blocco_generico]…
  |    '--[sottosezione…]
  |         |--faqh3 [id="ancora"] [bookmark="segnalibro"]
  |         |    '--testo_lineare
  |         |--qh3 [id="ancora"] [bookmark="segnalibro"]
  |         |    '--testo_lineare
  |         '--[blocco_generico]…
  '--[endofchapter]
```

Figura u65.27. Schema parziale di un capitolo contenente un questionario.

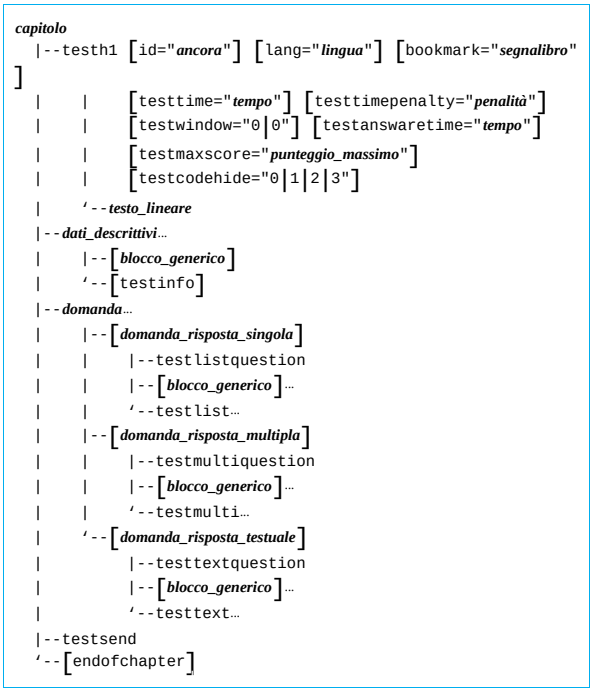


Tabella u65.28. Dichiarazione dei titoli di capitoli e di sezioni speciali.

Elemento	Descrizione
`slideh1` [`id="`*ancora*`"`] [`lang="…"`] [`bookmark="…"`]	Titolo della diapositiva. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`lang`**' consente di specificare la lingua del capitolo; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`sheeth1` [`id="`*ancora*`"`] [`lang="…"`] [`bookmark="…"`]	Titolo della scheda sintetica. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`lang`**' consente di specificare la lingua del capitolo; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`testh1` [`id="`*ancora*`"`] [`lang="…"`] [`bookmark="…"`] [`testtime="…"`] [`testtimepenalty="…"`] [`testwindow="0|1"`] [`testanswaretime="`*tempo*`"`] [`testmaxscore="`*massimo*`"`] [`testcodehide="`*n*`"`]	Titolo del questionario. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`lang`**' consente di specificare la lingua del capitolo; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF; l'attributo '**`testtime`**' consente di indicare il tempo massimo in secondi; '**`testtimepenalty`**' permette di specificare la penalità da sottrarre al punteggio per ogni secondo di ritardo; '**`testwindow`**' consente di far eseguire la verifica in una finestra priva di menù e di icone; '**`testanswaretime`**' consente di stabilire il tempo a disposizione per la stampa del risultato; '**`testmaxscore`**' serve a indicare ad Alml qual è il punteggio massimo che può produrre la verifica; '**`testmaxscore`**' serve a indicare ad Alml qual è il punteggio massimo che può produrre la verifica; '**`testcodehide`**' consente di rendere difficilmente interpretabile il codice HTML e JavaScript, attribuendo un valore intero maggiore di zero.

Elemento	Descrizione
`faqh2` [`id="`*ancora*`"`] [`lang="…"`]	Titolo del gruppo di domande e risposte. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`faqh3` [`id="`*ancora*`"`] [`lang="…"`]	Domanda a cui segue una risposta. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`qh2` [`id="`*ancora*`"`] [`lang="…"`]	Titolo di un gruppo di domande. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.
`qh3` [`id="`*ancora*`"`] [`lang="…"`]	Domanda. L'attributo '**`id`**' consente di specificare un'ancora di riferimento; l'attributo '**`bookmark`**' consente di specificare un segnalibro alternativo per la composizione in formato PDF.

Gli elementi '**`slideh1`**', '**`sheeth1`**' e '**`testh1`**' si usano al posto di un capitolo normale. La differenze più importante rispetto all'elemento '**`h1`**', sta nel fatto che non possono contenere altre suddivisioni in sezioni. Si osservi che, pur non avendo modo di controllare la dimensione del contenuto, è bene che ogni diapositiva e ogni scheda occupi una sola pagina nella composizione per la stampa, mentre nel caso di un questionario di verifica, non esiste questo problema estetico.

L'elemento '**`faqh2`**' va usato al posto di '**`h2`**', all'interno di un capitolo normale. Permette di introdurre un gruppo di domande e risposte, precedendole eventualmente da qualche blocco di testo introduttivo. L'elemento '**`qh2`**' è simile a '**`faqh2`**' e va usato quando le domande che raggruppa non hanno propriamente il senso di una «faq».

L'elemento '**`faqh3`**' serve a contenere il testo di una domanda, anche se potrebbe essere più lungo di un titolo normale. Il testo viene rappresentato in modo evidenziato, ma non tanto quanto un elemento '**`h3`**' normale. Dopo l'elemento '**`faqh3`**' ci si aspetta di trovare la risposta alla domanda. L'elemento '**`qh3`**' è simile a '**`faqh3`**' e va usato quando la domanda non ha il senso di una «faq».

Nel capitolo u77 è descritto meglio come realizzare delle diapositive, mentre nel capitolo u73 è descritto come realizzare un questionario.

Documento multilingua

L'attributo '**`lang`**' viene indicato normalmente nell'elemento '**`alml`**' per definire il linguaggio complessivo del lavoro, ma il linguaggio può essere modificato nell'ambito dei volumi, delle parti o dei capitoli. Per questo, l'attributo '**`lang`**' può essere usato anche negli elementi '**`tomeheading`**', '**`h0`**' e '**`h1`**', con lo scopo di fare riferimento al volume, alla parte o al capitolo a cui questi titoli si riferiscono. Pertanto, si osservi che l'attributo '**`lang`**' attribuisce il valore della scelta linguistica a tutto il volume, alla parte o al capitolo relativo, senza limitarsi all'ambito dell'elemento che ne delimita il titolo.

Un volume, una parte o un capitolo che non abbiano la definizione esplicita di un linguaggio, ereditano la definizione del livello gerarchicamente precedente.

La motivazione più importante per la quale è stato introdotto questo attributo nella dichiarazione dei volumi, delle parti e dei capitoli, sta nel fatto che così la composizione in HTML genera file con intestazioni adeguate, anche per l'indicizzazione delle informazioni.

La sigla della lingua va attribuita secondo lo standard ISO 639 (tabella 13.4). Se non è stata prevista la traduzione dei termini relativi alla composizione nella lingua richiesta, questi si ottengono in inglese.

L’esempio seguente mostra la dichiarazione esplicita di un capitolo che è da considerare in lingua inglese:

```
<h1 lang="en">Here I am</h1>
```

La definizione del volume, della parte o del capitolo viene adattata alla lingua, solo se questa non è stata modificata attraverso gli elementi ‘**tomedefinition**’, ‘**partdefinition**’ e ‘**chapterdefinition**’, descritti più avanti in questo capitolo.

Cambiamento temporaneo del linguaggio

Quando si inserisce un testo di un linguaggio che non appartiene all’Europa occidentale, può essere necessario selezionare il linguaggio per ottenere una composizione corretta. Pertanto, oltre alla selezione del linguaggio all’inizio dei volumi, delle parti e dei capitoli, è possibile modificare il linguaggio di un blocco di testo o di una porzione lineare, rispettivamente con gli elementi ‘**div**’ e ‘**span**’. Gli esempi seguenti mostrano l’uso di entrambi per ottenere la composizione per la stampa di alcune lettere in cirillico:

```
<div lang="ru">

    <p>&acy; &Acy; &bcy; &Bcy; &vcy; &Vcy; &gcy; &Gcy; &dcy; &Dcy;
    &iecy; &IEcy; &iocy; &IOcy; &zhcy; &ZHcy; &zcy; &Zcy;...</p>

</div>
```

а А б Б в В г Г д Д е Е ё Ё ж Ж з З...

```
<p>Bla bla bla: <span lang="ru">&acy; &Acy; &bcy; &Bcy; &vcy; &Vcy;
&gcy; &Gcy; &dcy; &Dcy; &iecy; &IEcy; &iocy; &IOcy; &zhcy; &ZHcy; &zcy;
&Zcy;...</span> bla bla bla.</p>
```

Bla bla bla: а А б Б в В г Г д Д е Е ё Ё ж Ж з З... bla bla bla.

Può succedere che il cambiamento di linguaggio crei «confusione» a LaTeX, che viene usato per ottenere la composizione da stampare. Si possono osservare degli errori inspiegabili nel file ‘`.log`’ generato durante il procedimento di composizione, in corrispondenza di vocali accentate:

```
! Missing \endcsname inserted.
<to be read again>
                   \global
l.16570 cui corrisponde l’entit\‘a
                                    parametrica
The control sequence marked <to be read again> should
not appear between \csname and \endcsname.
```

Se questo avviene successivamente a un testo scritto con un linguaggio particolare (proprio come si verifica in questa spiegazione), si può tentare di dichiarare nuovamente il linguaggio con un elemento ‘**div**’, oppure ‘**span**’:

```
<p>Bla bla bla: <span lang="ru">&acy; &Acy; &bcy; &Bcy; &vcy; &Vcy;
&gcy; &Gcy; &dcy; &Dcy; &iecy; &IEcy; &iocy; &IOcy; &zhcy; &ZHcy; &zcy;
&Zcy;...</span> bla bla bla.</p>

<div lang="it">

    <p>Bla bla bla... perché, poiché, c’è,... bla bla bla.</p>

</div>
```

Definizione alternativa della suddivisione del documento

Alml è pensato per la realizzazione di documenti di grandi dimensioni. In questo senso, la sua struttura normale è quella di un libro, articolato in capitoli che si possono raggruppare in parti e volumi. Queste suddivisioni prevedono una denominazione attribuita automaticamente, corrispondente a «capitolo», «parte» e «volume»; eventualmente, se tale struttura va definita invece attraverso termini differenti, si possono sostituire le definizioni delle suddivisioni con altre più appropriate.

Per questo si usano gli elementi ‘**chapterdefinition**’, ‘**partdefinition**’ e ‘**tomedefinition**’, all’interno delle informazioni amministrative. L’esempio seguente dovrebbe permettere di comprendere il problema; per la precisione si tratta di una rivista telematica ipotetica:

```
<head>
    <admin>
        <description>Rivista di informatica libera</description>
        <keywords>informatica libera, software libero</keywords>
        <chapterdefinition>articolo</chapterdefinition>
        <partdefinition>numero</partdefinition>
        <tomedefinition>anno</tomedefinition>
    </admin>
    <title>RIL, rivista di informatica libera</title>
    <author>Pinco Pallino &lt;pinco.pallino@brot.dg&gt;</author>
    <date>2011.11.11</date>
    <legal>
        <p>Copyright &copy; Pinco Pallino, &lt;pinco.pallino@brot.dg&gt;</p>
    </legal>
    <maincontents levels="2">Table of contents</maincontents>
</head>
```

Si può osservare che le parole «articolo», «numero» e «anno», sono state inserite usando lettere minuscole e in forma singolare. Ciò è necessario, perché l’iniziale maiuscola viene ottenuta automaticamente quando opportuno; inoltre, questi termini vengono usati sempre quando si fa riferimento a un solo oggetto.

La numerazione dei volumi, delle parti e dei capitoli è indipendente, per cui non ci si può aspettare che al cambio di un volume o di una parte, i capitoli riprendano la numerazione a partire da uno.

[1] Qui si intendono sezioni a qualsiasi livello, compresi i capitoli, le parti e i volumi.

Elementi interni alle righe

Numeri 430
Tastiera, menù e codice ASCII 431
Indirizzi di posta elettronica 432

Esistono due gruppi fondamentali di elementi: contenitori a blocco e contenitori lineari. Nel primo caso si possono immaginare dei rettangoli che contengono qualcosa, mentre nel secondo si tratta generalmente di sequenze di caratteri che scorrono e vanno a capo quando serve. Il caso tipico di elemento che costituisce un blocco è il «paragrafo», '**p**', che a sua volta contiene componenti lineari, mentre il caso tipico di elemento che può essere inserito esclusivamente in un contesto lineare è l'enfatizzazione, '**em**'. La tabella successiva riepiloga gli elementi comuni che riguardano inserzioni all'interno della riga.

Tabella u66.1. Elementi inseriti all'interno delle righe.

Elemento	Descrizione
`em`	Delimita un testo che deve essere reso in modo enfatizzato normale.
`strong`	Delimita un testo che deve essere reso in modo enfatizzato rafforzato.
`big`	Delimita un testo che deve apparire relativamente più grande.
`small`	Delimita un testo che deve apparire relativamente più piccolo.
`acronym`	Delimita un acronimo.
`dacronym`	Delimita la descrizione di un acronimo.
`kbd`	Delimita un testo che rappresenta la pressione di un tasto o di una combinazione di tasti.
`vkbd`	Delimita un testo che rappresenta la selezione virtuale di un tasto o di una combinazione di tasti.
`kp`	Delimita un testo che rappresenta la pressione di un tasto o di una combinazione di tasti della porzione numerica della tastiera.
`asciicode`	Delimita un testo che rappresenta un codice ASCII.
`button`	Delimita un testo che rappresenta la selezione di un bottone grafico.
`menuitem`	Delimita un testo che rappresenta la voce di un menù.
`code`	Codice: delimita un testo con la stessa funzione dell'elemento '**CODE**' di HTML.
`samp`	Stringa: delimita un testo con la stessa funzione dell'elemento '**SAMP**' di HTML.
`file`	Delimita il testo che rappresenta il percorso di un file o di una directory.
`dfn`	Delimita un testo da intendere come definizione.
`strdfn`	Delimita un testo da intendere come definizione espressa in una lingua straniera.

Elemento	Descrizione
`special special="`***nome***`"`	Delimita un testo che ha un significato speciale e appartiene a un gruppo di termini definito dal nome assegnato all'attributo.
`sup` `pwr`	Questi due elementi, delimitano un testo che deve essere elevato ad apice. Nel secondo caso, si intende che debba trattarsi di una potenza.
`sub`	Delimita un testo che deve essere abbassato a pedice.
`email`	Delimita un testo da trattare come indirizzo di posta elettronica da mascherare. Si usa delimitando il contenuto in una sezione marcata di tipo '**`CDATA`**' e serve a mascherare gli indirizzi ai sistemi automatici di raccolta di tali informazioni.
`num`	Delimita un numero normale, composto da cifre numeriche, punto o virgola e segno iniziale.
`exa`	Delimita un numero in base sedici.
`dec`	Delimita un numero in base dieci.
`oct`	Delimita un numero in base otto.
`bin`	Delimita un numero in base due.

Nelle sezioni successive viene approfondito l'uso di alcuni di questi elementi.

Numeri

La rappresentazione uniforme di valori numerici, specie quando si opera spesso con basi di numerazione insolite, diventa un aspetto delicato. Alml prevede alcuni elementi da utilizzare all'interno delle righe per delimitare valori numerici, eventualmente con basi di numerazione particolari, come si vede nella tabella successiva:

Tabella u66.2. Elementi inseriti all'interno delle righe per la rappresentazione uniforme di valori numerici.

Elemento	Descrizione
`num`	Delimita un numero normale, composto da cifre numeriche, punto o virgola e segno iniziale.
`exa`	Delimita un numero in base sedici. Può contenere anche gli elementi '**`var`**', '**`em`**' e '**`strong`**'.
`dec`	Delimita un numero in base dieci. Può contenere anche gli elementi '**`var`**', '**`em`**' e '**`strong`**'.
`oct`	Delimita un numero in base otto. Può contenere anche gli elementi '**`var`**', '**`em`**' e '**`strong`**'.
`bin`	Delimita un numero in base due. Può contenere anche gli elementi '**`var`**', '**`em`**' e '**`strong`**'.

Il caso dell'elemento '**`num`**' è speciale: si fa riferimento a un numero in base dieci in cui non si mostra la base di numerazione, ma si usa una modalità di rappresentazione standard. Per questa ragione, il numero in questione deve essere inserito come previsto, utilizzando la virgola o il punto come separatore della parte decimale,[1] aggiungendo il segno all'inizio, se necessario, senza usare altri spazi o altri caratteri. Il numero viene elaborato separando le cifre a terne.

Per quanto riguarda gli altri elementi per la rappresentazione di valori numerici, a seconda del tipo di composizione si utilizza un modo diverso per mostrare la base di numerazione. Si osservi l'esempio seguente:

```
<p>Il numero <num>1234</num> si può esprimere secondo varie basi di
numerazione: <bin>10011010010</bin>, oppure <oct>2322</oct>, oppure
<dec>1234</dec>, oppure <exa>4D2</exa>.</p>
```

Ecco il risultato dopo la composizione:

Il numero 1234 si può esprimere secondo varie basi di numerazione: 10011010010_{2}, oppure 2322_{8}, oppure 1234_{10}, oppure $4D2_{16}$.

Tastiera, menù e codice ASCII

Alml prevede diversi elementi per indicare l'interazione con la tastiera, con i programmi e per individuare dei codici ASCII speciali. Si distingue tra tastiera reale, tastiera virtuale, codici ASCII, bottoni grafici e voci di menù dei programmi.

Tabella u66.5. Elementi che riguardano l'uso della tastiera, l'individuazione di codici ASCII e l'uso dei programmi.

Elemento	Descrizione
`kbd`	Delimita un testo che rappresenta la pressione di un tasto o di una combinazione di tasti.
`vkbd`	Delimita un testo che rappresenta la selezione virtuale di un tasto o di una combinazione di tasti.
`kp`	Delimita un testo che rappresenta la pressione di un tasto o di una combinazione di tasti della porzione numerica della tastiera.
`asciicode`	Delimita un testo che rappresenta un codice ASCII.
`button`	Delimita un testo che rappresenta la selezione di un bottone grafico.
`menuitem`	Delimita un testo che rappresenta la voce di un menù.

Segue la descrizione di alcuni esempi.

- ```
  Attraverso le combinazioni di tasti
  <kbd>Ctrl Alt <kp>+</kp></kbd> e
  <kbd>Ctrl Alt <kp>-</kp></kbd> si può controllare la
  risoluzione dello schermo grafico.
  ```

  Attraverso le combinazioni di tasti [*Ctrl Alt* ⊞] e [*Ctrl Alt* ⊟] si può controllare la risoluzione dello schermo grafico.

- ```
  <p>La combinazione virtuale <vkbd>Meta_bracketleft</vkbd> si ottiene
  come <kbd>Alt [</kbd>, che in pratica può essere ottenuta come
  <kbd>Alt AltGr è</kbd>, oppure
  <kbd>Alt AltGr 8</kbd>. Naturalmente, quando il simbolo da
  combinare si trova nel quarto livello, occorre inserire nella
  combinazione reale anche il tasto <kbd>Maiuscole</kbd>.</p>
  ```

  La combinazione virtuale <*Meta_bracketleft*> si ottiene come [*Alt [*], che in pratica può essere ottenuta come [*Alt AltGr è*], oppure [*Alt AltGr 8*]. Naturalmente, quando il simbolo da combinare si trova nel quarto livello, occorre inserire nella combinazione reale anche il tasto [*Maiuscole*].

- ```
  In pratica, si deve considerare che le tastiere di un elaboratore
  comune si possono riconfigurare; pertanto, per fare un esempio, scrivere
  <asciicode>^a</asciicode> significa fare riferimento al codice ASCII
  <exa>01</exa>, pari a <asciicode>SOH</asciicode>, ma non è detto,
  necessariamente, che per ottenere questo codice si debba premere sulla
  tastiera di oggi una combinazione del tipo <kbd>Ctrl a</kbd>.
  ```
  ```

In pratica, si deve considerare che le tastiere di un elaboratore comune si possono riconfigurare; pertanto, per fare un esempio, scrivere <^a> significa fare riferimento al codice ASCII 01_{16}, pari a <*SOH*>, ma non è detto, necessariamente, che per ottenere questo codice si debba premere sulla tastiera di oggi una combinazione del tipo [*Ctrl a*].

-
```
<p>Si può salvare il documento selezionando la voce <menuitem>Save
as</menuitem> dal menù
<menuitem>File</menuitem>, specificando poi il nome del file, che ha
preferibilmente l'estensione <file>.lyx</file>.</p>
```

Si può salvare il documento selezionando la voce *Save as* dal menù *File*, specificando poi il nome del file, che ha preferibilmente l'estensione '`.lyx`'.

-
```
<p>Si può scegliere se prelevare semplicemente le tracce, generando file
di tipo WAV-RIFF, con il pulsante grafico <button>Rip only</button>,
oppure si può ottenere direttamente la conversione in formato MP3, con
il pulsante grafico <button>Rip+Encode</button>.</p>
```

Si può scegliere se prelevare semplicemente le tracce, generando file di tipo WAV-RIFF, con il pulsante grafico RIP ONLY, oppure si può ottenere direttamente la conversione in formato MP3, con il pulsante grafico RIP+ENCODE.

Indirizzi di posta elettronica

Per evitare di favorire l'individuazione di indirizzi di posta elettronica nei documenti pubblicati per la consultazione in linea, occorre camuffare questi indirizzi in qualche modo. Per evitare di dovervi provvedere a mano, esiste l'elemento '**`email`**', che va usato come nell'esempio seguente:

```
<p>Tizio Tizi, raggiungibile all'indirizzo
<email><![CDATA[tizio@brot.dg]]></email>, ha scritto...</p>
```

Ecco il risultato visibile nella composizione:

Tizio Tizi, raggiungibile all'indirizzo *tizio(at)brot·dg*, ha scritto...

Si osservi che l'indirizzo di posta elettronica va indicato racchiuso in una sezione marcata di tipo '**`CDATA`**', esattamente come si vede nell'esempio appena mostrato.

[1] Il segno meno, va indicato con il trattino normale.

Blocchi comuni

Elenchi e simili 433
Testo letterale o quasi 435
Sezioni marcate «CDATA» e spazi 439
Modelli sintattici 439
Comandi 441

In questo capitolo vengono descritti i componenti più comuni che si comportano come blocchi, assieme a elementi accessori a questi, anche se riguardano un contesto lineare. Nelle sezioni successive non viene menzionato l'elemento '**p**', pur essendo questo il blocco più importante:

Figura u67.1. L'elemento '**p**' costituisce un blocco che contiene dei componenti lineari e non prevede alcun attributo.

```
p
'--componenti_lineari
```

Elenchi e simili

Gli elenchi di Alml sono molto semplici. Si tratta dei soliti elenchi puntati, numerati e descrittivi. Questi si comportano in modo molto simile all'HTML; la differenza sostanziale sta nel fatto che il contenuto delle voci è composto da uno o più blocchi di testo, mentre in HTML è consentita anche la presenza di righe pure e semplici.

Figura u67.2. Elenchi descrittivi.

```
dl
'--elemento_dell'elenco…
      |--dt
      |    '--componenti_lineari
      '--dd
           '--blocco…
```

Figura u67.3. Elenchi numerati o puntati.

```
ol|ul
 '--li…
     '--blocco…
```

Tabella u67.4. Elenchi.

Elemento	Descrizione
`dl`	Elenco descrittivo.
`dt`	Termine descrittivo di un elenco.
`dd`	Descrizione di una voce di un elenco descrittivo.
`ol`	Elenco numerato.
`ul`	Elenco puntato.
`li`	Elemento di un elenco numerato o puntato.

Segue la descrizione di alcuni esempi. Si comincia con un elenco puntato, suddiviso in sottoelenchi:

```
<p>Il documento si articola in:</p>
<ul>
<li>
    <p>volumi (o tomi)</p>
    <p>quando il documento è molto grande</p>
</li>
<li>
    <p>parti</p>
    <p>quando il volume richiede una suddivisione degli argomenti ben
    strutturata</p>
</li>
<li>
    <p>capitoli</p>
    <p>i capitoli, a loro volta, si articolano in:</p>
    <ul>
    <li>
        <p>sezioni</p>
    </li>
    <li>
        <p>sottosezioni</p>
    </li>
    <li>
        <p>sotto-sottosezioni</p>
    </li>
    </ul>
</li>
</ul>
```

Ecco come si presenta questo elenco:

Il documento si articola in:

- volumi (o tomi)
  quando il documento è molto grande
- parti
  quando il volume richiede una suddivisione degli argomenti ben strutturata
- capitoli
  i capitoli, a loro volta, si articolano in:
  - sezioni
  - sottosezioni
  - sotto-sottosezioni

Segue lo stesso esempio, utilizzando elenchi numerati:

```
<p>Il documento si articola in:</p>
<ol>
<li>
    <p>volumi (o tomi)</p>
    <p>quando il documento è molto grande</p>
</li>
<li>
    <p>parti</p>
    <p>quando il volume richiede una suddivisione degli argomenti ben
    strutturata</p>
</li>
<li>
    <p>capitoli</p>
    <p>i capitoli, a loro volta, si articolano in:</p>
    <ol>
    <li>
        <p>sezioni</p>
    </li>
    <li>
        <p>sottosezioni</p>
    </li>
    <li>
        <p>sotto-sottosezioni</p>
    </li>
    </ol>
</li>
</ol>
```

Ecco come si presenta:

Il documento si articola in:

1. volumi (o tomi)
   quando il documento è molto grande
2. parti
   quando il volume richiede una suddivisione degli argomenti ben strutturata
3. capitoli
   i capitoli, a loro volta, si articolano in:
   (a) sezioni
   (b) sottosezioni
   (c) sotto-sottosezioni

Segue un esempio per l'uso dell'elenco descrittivo:

```
<dl>
<dt><strong>volumi</strong></dt>
<dd>
    <p>Un documento di grandi dimensioni va suddiviso in volumi (o
    tomi).</p>
</dd>
<dt><strong>parti</strong></dt>
<dd>
    <p>Quando un volume richiede una suddivisione degli argomenti ben
    strutturata, va suddiviso in parti.</p>
</dd>
<dt><strong>capitoli</strong></dt>
<dd>
    <p>Un volume di piccole dimensioni o una parte, vanno suddivisi
    in capitoli. A sua volta, il capitolo si suddivise in sezioni
    fino a tre livelli ulteriori.</p>
    <dl>
    <dt><strong>sezioni</strong></dt>
    <dd>
        <p>Le sezioni sono la suddivisione principale dei capitoli.</p>
    </dd>
    <dt><strong>sottosezioni</strong></dt>
    <dd>
        <p>Le sezioni si suddividono in sottosezioni.</p>
    </dd>
    <dt><strong>sotto-sottosezioni</strong></dt>
    <dd>
        <p>Le sottosezioni si suddividono in sotto-sottosezioni
        e non sono previsti altri livelli inferiori.</p>
    </dd>
    </dl>
</dd>
</dl>
```

Ecco come si mostra l'elenco descrittivo:

volumi
Un documento di grandi dimensioni va suddiviso in volumi (o tomi).

parti
Quando un volume richiede una suddivisione degli argomenti ben strutturata, va suddiviso in parti.

capitoli
Un volume di piccole dimensioni o una parte, vanno suddivisi in capitoli. A sua volta, il capitolo si suddivise in sezioni fino a tre livelli ulteriori.

sezioni
Le sezioni sono la suddivisione principale dei capitoli.

sottosezioni
Le sezioni si suddividono in sottosezioni.

sotto-sottosezioni
Le sottosezioni si suddividono in sotto-sottosezioni e non sono previsti altri livelli inferiori.

Testo letterale o quasi

L'inclusione di testo letterale in un sorgente SGML è sempre un problema. Alml prevede tre ambienti diversi: '**`verbatimpre`**', '**`asciiart`**' e '**`pre`**'. Nei primo due casi si può scrivere senza alcuna preoccupazione, tranne per il fatto che il testo va inserito in una sezione marcata di tipo '**`CDATA`**'; nel terzo caso invece, è necessario comportarsi come nel testo normale, utilizzando le entità standard quando servono, potendo includere anche gran parte degli elementi che rappresentano un'inserzione all'interno di una riga. In tutti i casi vengono rispettate le interruzioni di riga.

```
<verbatimpre>
<![CDATA[
uno
   &
    due
]]>
</verbatimpre>
```

```
<pre>
uno
   &amp;
    due
</pre>
```

I due esempi portano allo stesso risultato:

```
uno
   &
    due
```

Gli elementi ‘**verbatimpre**’ e ‘**pre**’ possono anche essere bordati e numerati. L’esempio seguente mostra l’uso dell’elemento ‘**verbatimpre**’, dove le righe del suo contenuto devono essere numerate a partire dal numero uno:

```
<verbatimpre numbering="1">
<![CDATA[
drwxr-xr-x    2 root     root         4096 2003-01-17 15:47 bin
drwxr-xr-x    3 root     root         4096 2003-01-28 16:18 boot
drwxr-xr-x    1 root     root            0 1970-01-01 01:00 dev
drwxr-xr-x  139 root     root         8192 2003-01-30 16:47 etc
drwxrwsr-x   17 root     staff        4096 2003-01-19 22:01 home
drwxr-xr-x    6 root     root         4096 2003-01-11 15:26 lib
drwxr-xr-x    2 root     root        16384 2000-12-15 14:49 lost+found
drwxr-xr-x  311 root     root         8192 2003-01-22 16:36 mnt
dr-xr-xr-x   89 root     root            0 2003-01-30 14:30 proc
drwxr-xr-x   15 root     root         4096 2003-01-30 16:32 root
drwxr-xr-x    2 root     root         4096 2003-01-10 16:04 sbin
drwxrwxrwt    5 root     root       176128 2003-01-30 17:45 tmp
drwxr-xr-x   15 root     root         4096 2003-01-04 11:06 usr
drwxr-xr-x   16 root     root         4096 2002-10-27 18:25 var
]]>
</verbatimpre>
```

Ecco cosa si ottiene:

```
1     drwxr-xr-x    2 root     root         4096 2003-01-17 15:47 bin
2     drwxr-xr-x    3 root     root         4096 2003-01-28 16:18 boot
3     drwxr-xr-x    1 root     root            0 1970-01-01 01:00 dev
4     drwxr-xr-x  139 root     root         8192 2003-01-30 16:47 etc
5     drwxrwsr-x   17 root     staff        4096 2003-01-19 22:01 home
6     drwxr-xr-x    6 root     root         4096 2003-01-11 15:26 lib
7     drwxr-xr-x    2 root     root        16384 2000-12-15 14:49 lost+found
8     drwxr-xr-x  311 root     root         8192 2003-01-22 16:36 mnt
9     dr-xr-xr-x   89 root     root            0 2003-01-30 14:30 proc
10    drwxr-xr-x   15 root     root         4096 2003-01-30 16:32 root
11    drwxr-xr-x    2 root     root         4096 2003-01-10 16:04 sbin
12    drwxrwxrwt    5 root     root       176128 2003-01-30 17:45 tmp
13    drwxr-xr-x   15 root     root         4096 2003-01-04 11:06 usr
14    drwxr-xr-x   16 root     root         4096 2002-10-27 18:25 var
```

L’esempio seguente mostra l’uso dell’elemento ‘**pre**’, bordato:

```
<pre border="1">
uno
   &amp;
    due
</pre>
```

Ecco il risultato:

```
uno
   &
    due
```

È bene osservare che il testo inserito negli elementi ‘**verbatimpre**’, ‘**asciiart**’ e ‘**pre**’, dovrebbe essere limitato al primo gruppo di punti di codifica, corrispondente in pratica a ISO 8859-1. Diversamente si pongono due tipi di problemi: il carattere tipografico che si ottiene può essere differente e soprattutto può avere una spaziatura diversa; inoltre, in alcuni casi è indispensabile selezionare il linguaggio, cosa che non si può fare all’interno degli elementi ‘**verbatimpre**’ e ‘**asciiart**’, perché assolutamente letterali. Eventualmente, in caso di necessità si deve usare l’elemento ‘**pre**’, che invece consente l’inserimento dell’elemento ‘**span**’ al suo interno.

Tabella u67.16. Elementi SGML che riguardano la rappresentazione di testo preformattato.

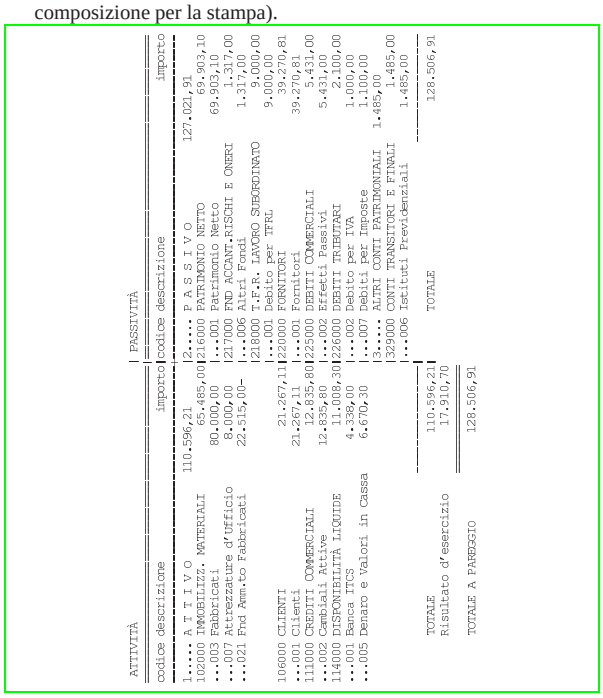
Elemento	Descrizione
`pre [width="`***n***`"] [border="0|1"] [ numbering="`***n***`"]`	Contiene testo lineare da mantenere impaginato come nel sorgente. L’attributo ‘**width**’ serve a richiede un certo numero di colonne; l’attributo ‘**border**’ serve a richiedere una cornice; l’attributo ‘**numbering**’ consente di numerare le righe a partire da un certo numero.
`pnewline`	Si tratta di un elemento vuoto da usare in un elemento ‘**pre**’, per spezzare le righe in modo visibile nella composizione finale.
`verbatimpre [width="`***n***`"] [border="0|1"] [numbering="`***n***`"] [ file="`***nome***`"]`	Contiene testo lineare letterale, da mantenere impaginato come nel sorgente. L’attributo ‘**width**’ serve a richiede un certo numero di colonne; l’attributo ‘**border**’ serve a richiedere una cornice; l’attributo ‘**numbering**’ consente di numerare le righe a partire da un certo numero; l’attributo ‘**file**’ consente di salvare una copia del contenuto in un file, in fase di composizione.
`asciiart [width="`***n***`"] [file="`***nome***`"] [rotated="0|1"]`	Contiene testo lineare letterale, da mantenere impaginato come nel sorgente. L’attributo ‘**width**’ serve a richiede un certo numero di colonne; l’attributo ‘**file**’ consente di salvare una copia del contenuto in un file, in fase di composizione; l’attributo ‘**rotated**’ consente di richiedere la rotazione del testo nella composizione per la stampa.

L’elemento ‘**asciiart**’ è diverso da ‘**verbatimpre**’, in quanto deve trovarsi inserito in un elemento ‘**object**’ (descritto in un altro capitolo); inoltre non può essere spezzato tra le pagine e appare sempre al centro della pagina (in orizzontale). Sempre nel caso della composizione stampata, l’elemento ‘**asciiart**’ può essere visualizzato ruotandolo di 90 gradi, così da poter sfruttare più spazio orizzontale. Segue un esempio il cui risultato nella composizione finale si vede nella figura u67.18:

```
<object split="0">
<asciiart width="96" rotated="1">
<![CDATA[
 ATTIVITÀ                                    | PASSIVITÀ
=============================================|=============================================
codice descrizione                  importo|codice descrizione                  importo
---------------------------------------------+---------------------------------------------
1..... A T T I V O            110.596,21     |2..... P A S S I V O           127.021,91
102000 IMMOBILIZZ. MATERIALI        65.485,00|216000 PATRIMONIO NETTO             69.903,10
...003 Fabbricati                   80.000,00|...001 Patrimonio Netto            69.903,10
...007 Attrezzature d'Ufficio        8.000,00|217000 FND ACCANT.RISCHI E ONERI       1.317,00
...021 Fnd Amm.to Fabbricati        22.515,00-|...006 Altri Fondi                  1.317,00
                                             |218000 T.F.R. LAVORO SUBORDINATO       9.000,00
                                             |...001 Debito per TFRL              9.000,00
106000 CLIENTI                      21.267,11|220000 FORNITORI                     39.270,81
...001 Clienti                      21.267,11|...001 Fornitori                    39.270,81
111000 CREDITI COMMERCIALI          12.835,80|225000 DEBITI COMMERCIALI             5.431,00
...002 Cambiali Attive              12.835,80|...002 Effetti Passivi              5.431,00
114000 DISPONIBILITÀ LIQUIDE        11.008,30|226000 DEBITI TRIBUTARI               2.100,00
...001 Banca ITCS                    4.338,00|...002 Debito per IVA               1.000,00
...005 Denaro e Valori in Cassa      6.670,30|...007 Debiti per Imposte           1.100,00
                                             |3..... ALTRI CONTI PATRIMONIALI   1.485,00
                                             |329000 CONTI TRANSITORI E FINALI       1.485,00
                                             |...006 Istituti Previdenziali       1.485,00
                                    ---------|                                  ----------
      TOTALE                        110.596,21|      TOTALE                       128.506,91
```

```
      Risultato d'esercizio          17.910,70
                                    ===========
      TOTALE A PAREGGIO             128.506,91
]]>
</asciiart>
</object>
```

Figura u67.18. Il risultato della composizione dell'esempio di utilizzo di '**asciiart**' ruotando il contenuto (per la sola composizione per la stampa).



Dalla descrizione fatta nella tabella u67.16, si può osservare che gli elementi '**verbatimpre**' e '**asciiart**' prevedono l'attributo '**file**', con lo scopo di salvare una copia del contenuto in un file, mentre si esegue la composizione per generare il risultato finale. Si legga il capitolo u72 a proposito degli allegati.

Gli elementi '**pre**', '**verbatimpre**', '**asciiart**' (compreso '**syntax**' che viene descritto nella sezione successiva), sono predisposti inizialmente per poter rappresentare 80 colonne di testo letterale, in una larghezza pari a quella normale del testo. In situazioni particolari può essere necessario ridurre (o ampliare) la dimensione dei caratteri nella composizione stampata, per consentire la rappresentazione di un testo più ampio orizzontalmente (o più breve, ma con caratteri più grandi). In questi casi, si può utilizzare l'attributo '**width**', assegnando la quantità di colonne che si desiderano. Seguono due esempi: nel primo caso si richiedono espressamente solo 60 colonne, in modo da ottenere un carattere un po' più grande del solito; nel secondo vengono richieste 90 colonne.

```
<pre width="60">
1234567890
          1234567890
                    1234567890
                              1234567890
                                        1234567890
                                                  1234567890
</pre>
```

```
<pre width="90">
1234567890
         1234567890
                  1234567890
                           1234567890
                                    1234567890
                                             1234567890
                                                      1234567890
                                                               1234567890
                                                                        1234567890
</pre>
```

In caso di necessità, se si vuole che la dimensione del carattere sia la stessa dell'ambiente in cui si trova l'elemento in questione, è sufficiente richiedere espressamente una larghezza pari a zero:

```
<pre width="0">
1234567890
          1234567890
                    1234567890
                              1234567890
                                        1234567890
                                                  1234567890
</pre>
```

Sezioni marcate «CDATA» e spazi

Si comprende intuitivamente che, gli spazi che si inseriscono all'interno di una sezione marcata di tipo CDATA hanno sempre valore. Esiste una sola eccezione, per cui i due esempi seguenti sono equivalenti:

```
<![CDATA[6t86546ftgiuy98yq435q0459823
2908430tfg76tr7852tg9j0090jh
432w7089hphg7t8680'09u76r78d]]>
```

```
<![CDATA[
6t86546ftgiuy98yq435q0459823
2908430tfg76tr7852tg9j0090jh
432w7089hphg7t8680'09u76r78d
]]>
```

Tuttavia, c'è la possibilità di fare degli errori senza rendersene conto, inserendo involontariamente degli spazi prima della fine della riga. L'esempio seguente riprende quello appena mostrato e mostra la conclusione della riga con il simbolo '¶':

```
<![CDATA[ ¶
6t86546ftgiuy98yq435q0459823¶
2908430tfg76tr7852tg9j0090jh¶
432w7089hphg7t8680'09u76r78d¶
]]>¶
```

In questo caso, si può osservare che c'è uno spazio tra l'inizio della sezione marcata e la conclusione della riga:

```
<![CDATA[ ¶
```

In questo modo, succede qualcosa che per chi non è esperto è impensabile: il contenuto della sezione marcata ha una riga iniziale vuota. In pratica, è come se il contenuto fosse semplicemente così:

```
 ¶
6t86546ftgiuy98yq435q0459823¶
2908430tfg76tr7852tg9j0090jh¶
432w7089hphg7t8680'09u76r78d¶
```

A seconda del significato del contenuto di una sezione marcata di questo tipo, può darsi che la riga iniziale aggiunta risulti ininfluente, oppure può far perdere qualunque significato a tali dati.

Modelli sintattici

In un documento a carattere tecnico-informatico, è essenziale la possibilità di indicare dei modelli sintattici. Alml prevede l'uso di un elemento simile a '**pre**', dedicato precisamente a questo scopo: '**syntax**'. Segue un esempio del suo utilizzo:

```
<syntax>
man <synsqb><var>n_sezione</var></synsqb> <var>nome</var>
</syntax>
```

Ecco come appare:

```
man [n_sezione] nome
```

All'interno di questo elemento si possono inserire altri elementi specifici per rappresentare i componenti della sintassi. Infatti, è necessario distinguere tra parole chiave, metavariabili e altre indicazioni. In generale, quello che si scrive normalmente deve essere inteso come un dato fisso, ovvero delle parole chiave o delle stringhe fisse. Per indicare un contenuto variabile si utilizza l'elemento '**var**' per delimitare la denominazione di un qualcosa di variabile (un'opzione o simile).

Altri elementi speciali servono a guidare la lettura della sintassi: '**synsqb**' delimita una parte della sintassi che va intesa come facoltativa e si traduce generalmente con delle parentesi quadre che, se possibile, si distinguono dal testo normale; '**syncub**' delimita una parte della sintassi che va intesa come un corpo unico e si traduce generalmente con delle parentesi graffe speciali; '**synverbar**' (elemento vuoto) indica un'alternativa e si rappresenta con una barra verticale; '**synellipsis**' (elemento vuoto) rappresenta dei puntini di sospensione particolari, diversi da quelli che si otterrebbero in modo normale; '**synstar**' (elemento vuoto) rappresenta una cosa simile all'asterisco secondo la shell tradizionale, da intendersi come sostituto di qualunque stringa. Nell'uso di questi elementi occorre sempre un po' di prudenza, tenendo conto dei tipi di composizione in cui non è possibile mostrare questi simboli in forme diverse dal normale.

Tabella u67.28. Elementi SGML che riguardano la rappresentazione di modelli sintattici.

Elemento	Descrizione
`syntax [width="n"] [border="0|1"] [split="0|1"] [numbering="n" ]`	Contiene un modello sintattico preformattato. L'attributo '**width**' consente di specificare una larghezza in colonne del modello; l'attributo '**border**' consente mettere un bordo attorno al modello; l'attributo '**split**' consente di rendere separabile il modello tra le pagine; l'attributo '**numbering**' consente di numerare le righe del modello a partire dal numero indicato.
`synsqb`	Delimita una porzione del modello sintattico, mostrando delle parentesi quadre (raggruppamento opzionale).
`syncub`	Delimita una porzione del modello sintattico, mostrando delle parentesi graffe (raggruppamento obbligatorio).
`synverbar`	È un elemento vuoto che mostra una barra verticale (alternativa).
`var`	Rappresenta una metavariabile sintattica.
`synellipsis`	È un elemento vuoto che mostra un'ellissi (ripetizione).
`snewline`	È un elemento vuoto che consente di spezzare una riga del modello, sottolineando il fatto che nella situazione a cui ci si riferisce, la riga dovrebbe essere continua.
`synstar`	È un elemento vuoto che consente di mostrare una stellina (un asterisco), da intendere come simbolo di qualunque cosa. In pratica, lo si intende come si farebbe per una shell POSIX, ma anche in contesti estranei alla digitazione di comandi del sistema operativo.

Si tenga in considerazione il fatto che gli elementi '**synsqb**', '**syncub**', '**synverbar**', '**synellipsis**' e '**var**', possono essere utilizzati anche al di fuori dell'elemento '**syntax**', in qualità di inserzioni normali nelle righe.

La riga di un modello sintattico che si estende troppo in orizzontale, può essere spezzata e ripresa inserendo l'elemento vuoto '**snewline**', in modo da ottenere una segnalazione evidente nella composizione finale, senza lasciare ambiguità. La stessa cosa, eventualmente, si può fare nell'elemento '**pre**', usando l'elemento vuoto '**pnewline**'. Si osservi l'esempio seguente che si riferisce a un modello sintattico:

```
<syntax border="1">
pippo --primo <synverbar> <snewline>--secondo <synverbar> --terzo
</syntax>
```

Ecco cosa si ottiene:

```
pippo --primo | ↩
↪--secondo | --terzo
```

Quando si usa un elemento come '**snewline**', '**pnewline**' o '**cnewline**', vicino a uno spazio orizzontale, è bene che lo spazio venga lasciato prima dell'inserzione dell'elemento stesso, senza eliminarlo, in modo da sottolinearne la presenza.

Comandi

I comandi che si impartiscono attraverso una riga di comando, possono essere rappresentati con l'elemento '**command**'. Si osservi l'esempio seguente:

```
<command><prompt>$ </prompt><type>ls -l</type><kbd>Invio</kbd></command>
```

Ecco come appare:

$ **ls -l**[*Invio*]

Nell'ambito dell'elemento '**command**' è quasi tutto facoltativo; tuttavia, l'invito, rappresentato dall'elemento '**prompt**', va messo per primo. Dopo l'elemento '**type**', che serve a delimitare il testo che viene inserito sulla riga di comando, è possibile anche specificare il tasto che serve a concludere la digitazione, come in questo caso, oppure se ne può fare a meno, lasciandolo sottinteso.

Il testo che viene restituito da un comando si rappresenta normalmente con l'elemento '**verbatimpre**' o '**pre**', contenuto in un elemento '**object**'.

A volte, si ha la necessità di rappresentare dei comandi piuttosto lunghi, che nella composizione stampata potrebbero risultare spezzati in modo imprevedibile e indesiderabile. È possibile indicare esplicitamente dove spezzare il comando, facendo in modo che nella composizione si intenda chiaramente questo fatto. Per questo si usa l'elemento vuoto '**cnewline**', che si inserisce all'interno di '**type**'.

Figura u67.32. Sintassi semplificata per l'uso dell'elemento '**command**'.

```
command
  |--prompt
  |    '--testo_lineare
  |--type
  |    '--[testo_lineare|cnewline]…
  '--[kbd|button]
```

Tabella u67.33. Elementi SGML che servono a rappresentare un comando.

Elemento	Descrizione
`command`	Comando da digitare.
`prompt`	Stringa dell'invito.
`type`	Digitazione del comando.

Elemento	Descrizione
`cnewline`	Elemento vuoto per continuare il comando a riga nuova.
`kbd`	Tasto o combinazione di tasti da premere.
`button`	Bottone o tasto grafico da selezionare.

Altri blocchi e componenti lineari particolari

Inserzioni particolari 443
Riquadri 444
Copia di porzioni del documento 446
Copia di piè di pagina 447
Copia di immagini 447

Dopo la descrizione di elementi di uso abbastanza semplice, conviene concentrare l'attenzione su altri elementi importanti con funzioni speciali.

Inserzioni particolari

Sono disponibili diversi elementi di importanza minore. Si tratta di '**`br`**', '**`hr`**', '**`newpage`**', '**`bottompage`**', '**`heightrequired`**' e '**`navlink`**'. I primi due emulano gli elementi corrispondenti dell'HTML, interrompendo una riga e inserendo una linea orizzontale rispettivamente.

L'elemento '**`newpage`**' richiede un salto pagina, se il tipo di composizione lo consente.

L'elemento '**`bottompage`**' serve per definire un gruppo di blocchi di testo da rappresentare nella parte bassa della pagina, nella composizione per la stampa. In pratica, si usa '**`bottompage`**' per delimitare informazioni legali nella seconda pagina relativa dei volumi:

```
<tomeheading>Bla bla bla</tomeheading>

<bottompage>
    <p>Copyright &copy; Pinco Pallino...</p>

    <p>Bla bla bla...</p>
</bottompage>
```

L'elemento '**`heightrequired`**' serve nella composizione per la stampa, a garantire che sia disponibile una certa quantità di spazio (un'altezza minima prima della fine della pagina), in mancanza del quale viene inserito un salto pagina. Questo elemento serve per rimediare agli errori di composizione che compaiono di tanto in tanto.

Tabella u68.2. Inserzioni varie.

Elemento	Descrizione
`br`	Elemento vuoto che manda a capo il testo, da usare in un contesto lineare.
`hr`	Elemento vuoto che inserisce una riga orizzontale di separazione. Può essere usato solo tra un blocco e l'altro.
`newpage`	Elemento vuoto che richiede un salto pagina, se il contesto lo consente. Può essere usato solo tra un blocco e l'altro.
`bottompage`	Elemento contenente blocchi che richiede una rappresentazione alla base della pagina nella composizione per la stampa.
`heightrequired height="`***`altezza`***`"`	Elemento vuoto che serve a richiedere espressamente la presenza di una certa quantità di spazio prima della fine della pagina. Si tratta evidentemente di un elemento da usare tra un blocco e l'altro. L'attributo '**`height`**' serve a specificare l'altezza minima richiesta.

Elemento	Descrizione
`navlink`	Elemento contenente una stringa da usare come riferimento alla pagina in cui viene collocato, per la navigazione HTML. Tale riferimento viene inserito in tutte le pagine HTML risultanti dalla composizione.

L'elemento '**`navlink`**' consente di aggiungere nella composizione HTML un riferimento ipertestuale fisso, in tutte le pagine, allo scopo di raggiungere facilmente la posizione in cui l'elemento stesso viene inserito. Si osservi l'esempio seguente:

```
<h1>
Indice analitico
</h1>

<navlink>indice analitico</navlink>

<printindex index="main">

</index>
```

Si tratta dell'inserimento dell'indice analitico, con l'aggiunta di un riferimento ipertestuale fisso nelle pagine della composizione HTML.

Figura u68.4. Esempio di una pagina HTML prodotta dalla composizione di un sorgente contenente un riferimento aggiuntivo per la consultazione, denominato '**`indice analitico`**'.

Capitolo 3. Standard

Attorno ai sistemi operativi che si rifanno al modello di Unix, si sono definiti degli standard importanti. Vengono qui annotati alcuni riferimenti a proposito di questi standard; tuttavia, si tenga presente che questo è sempre un campo in evoluzione e nulla è definitivo.

3.1 Linguaggio C

Il linguaggio C è quello su cui si basano i sistemi Unix; l'evoluzione dei sistemi Unix va di pari passo con quella del suo linguaggio.

- Brian W. Kernigham, Dennis M. Ritchie, *The C programming language*, prima edizione, Prentice-Hall 1978

  Questo è il primo documento che definisce il linguaggio C, per quello che oggi è noto come «K&R C», ovvero il linguaggio C di Kernigham e Ritchie. Di questo libro esiste una seconda edizione, del 1988, rivista secondo le convenzioni in corso di definizione dallo standard ANSI C.

  <http://cm.bell-labs.com/cm/cs/cbook/>).

[...]

Dovrebbe essere possibile fare riferimento a questa pagina anche con il nome standard.htm

Riquadri

Almlml consente di inserire nel documento dei riquadri, a cui si associa una numerazione separata rispetto alle sezioni, che eventualmente possono essere resi fluttuanti nel testo. Questi riquadri sono ottenuti con l'elemento '**`object`**'.

Figura u68.5. Sintassi semplificata per l'uso dei riquadri.

```
object [id="ancora"] [pos="fixed|float"]
  |    [sep="none|rule|border"] [split="0|1"]
  |    [printedfontsize="dimensione"]
  |--[caption]
  |      ' - - testo_lineare
  ' - - blocco...
```

L'elemento '**`object`**' può contenere una didascalia, delimitata dall'elemento '**`caption`**', che a sua volta contiene testo lineare; quindi può contenere blocchi di vario tipo, compresi dei blocchi speciali che possono apparire solo al suo interno (come nel caso dell'elemento '**`asciiart`**'). Nell'insieme, il riquadro può essere bordato o meno, può essere fisso o fluttuante, può essere separato tra le pagine oppure può essere un blocco unico. L'esempio seguente mostra un caso tipo:

```
<object sep="border" pos="float" id="a2-mio-riquadro-di-prova"
        printedfontsize="0.9em">
<caption>
    Riquadro <objectref>. Avvio di un disco esterno e ritardo nel
    kernel.
</caption>
    <p>Per l'avvio di nanoLinux installato in un disco USB è importante
    considerare che tra le opzioni del kernel deve essere prevista la
    presenza di <samp>setupdelay</samp>, a cui si assegna un numero
    intero che rappresenta un ritardo in secondi prima dell'innesto del
    file system principale. Questa opzione è presente solo nel kernel
    realizzato per nanoLinux e consente di avviare un disco USB senza
    bisogno di un disco RAM. Questa opzione è già presente nel file
    <file>boot/grub/menu.lst</file> per le voci riferite a dischi di
    questo tipo, con un ritardo di <num>5</num> s.</p>
</object>
```

Il risultato della composizione di questo esempio, si può osservare nel riquadro successivo:

Riquadro u68.8. Avvio di un disco esterno e ritardo nel kernel.

Per l'avvio di nanoLinux installato in un disco USB è importante considerare che tra le opzioni del kernel deve essere prevista la presenza di '**`setupdelay`**', a cui si assegna un numero intero che rappresenta un ritardo in secondi prima dell'innesto del file system principale. Questa opzione è presente solo nel kernel realizzato per nanoLinux e consente di avviare un disco USB senza bisogno di un disco RAM. Questa opzione è già presente nel file '`boot/grub/menu.lst`' per le voci riferite a dischi di questo tipo, con un ritardo di 5 s.

Si osservi che le opzioni definite attraverso gli attributi dell'elemento '**`object`**' non possono convivere sempre in tutte le condizioni. In particolare, se il riquadro viene bordato attraverso l'elemento '**`object`**' stesso, non è possibile ottenere che il contenuto si possa separare tra le pagine. A questo proposito, si può osservare invece che elementi come '**`pre`**', '**`verbatimpre`**' e '**`syntax`**', si possono bordare e separare tra le pagine, ma in tal caso, se vengono inseriti in un elemento '**`object`**', questo deve risultare non bordato, lasciando il compito della bordatura agli elementi contenuti.

Tabella u68.9. Riquadri.

Elemento	Descrizione
`object` [`id="`***`ancora`***`"`] [`pos="fixed`|`float"`] [`sep="none`|`rule`|`border"`] [`split="0`|`1"`] [`printedfontsize="`***`dimensione`***`"`]	Involucro di un riquadro. L'attributo '**`id`**' consente di mette un'ancora di riferimento; l'attributo '**`pos`**' consente di rendere fluttuante il riquadro; l'attributo '**`sep`**' consente di definire un bordo esterno; l'attributo '**`printedfontsize`**' consente di definire la dimensione del carattere normale da usare nel riquadro; l'attributo '**`split`**' consente di stabilire se il riquadro debba rimanere unito o possa essere suddiviso.
`caption`	Contiene la didascalia, in forma di testo lineare.

Il corpo del carattere «normale» che si inserisce all'interno di un riquadro, può essere controllato con l'attributo '**`printedfontsize`**', oppure, in modo generale, nell'intestazione con un elemento '**`printedfontsize`**', come nell'esempio seguente:

```
<head>
    <admin>
        …
        <printedfontsize type="object">3,5mm</printedfontsize>
        …
    </admin>
    …
</head>
```

Se non si indica questa informazione, il carattere viene ridotto leggermente rispetto a quello del corpo normale del testo; se invece si vuole mantenere un carattere uguale a quello del contesto esterno, basta usare l'attributo '**printedfontsize**' indicando una dimensione pari a un quadratone, come nell'esempio seguente:

```
<object sep="border" pos="float" printedfontsize="1em">
    …
    …
</object>
```

Non si deve confondere il riquadro costituito dall'elemento '**object**' con la cornice dell'elemento '**frame**'. L'elemento '**frame**' (sezione u0.2) serve per mettere in evidenza una nota **breve**, mentre l'elemento '**object**' delimita un contenuto autonomo che potrebbe avere una didascalia.

Copia di porzioni del documento

Alcune porzioni del documento che si scrive con Alml, possono essere copiate in posizioni successive. Ciò si ottiene con gli elementi '**copy**' e '**paste**'.

Gli elementi '**copy**' e '**paste**' possono essere usati sia in un contesto che richiede l'uso di blocchi, sia quando il contesto è lineare; di conseguenza, il loro contenuto può essere fatto di blocchi o di testo lineare.

Tabella u68.12. Copia di porzione del documento.

Elemento	Descrizione
`cut cut="`***area_di_memoria***`"`	Delimita la porzione di documento da accumulare nell'area denominata come indicato con l'attributo '**copy**'. Il contenuto dell'elemento non appare nella composizione finale.
`copy copy="`***area_di_memoria***`"`	Delimita la porzione di documento da accumulare nell'area denominata come indicato con l'attributo '**copy**'.
`paste paste="`***area_di_memoria***`"`	Si tratta di un elemento vuoto che inserisce in quel punto quanto accumulato nella voce indicata con l'attributo '**paste**'.

L'esempio seguente serve ad accumulare alcuni paragrafi in un'area di memoria denominata '**commenti**':

```
<p>Bla bla bla bla...</p>
<copy copy="commenti">
    <p>Che sciocchezze che si scrivono negli esempi...</p>
    <p>Cosa si può aggiungere di più?</p>
</copy>
<p>Bla bla bla.</p>
```

Nell'esempio successivo, si recupera quanto accumulato in precedenza nell'area di memoria '**commenti**':

```
<p>Ecco i commenti fatti fino a questo punto:</p>
<frame>
    <paste paste="commenti">
</frame>
```

Si osservi che l'elemento '**copy**' accumula blocchi o testo lineare in memoria, ma questi rimangono visibili normalmente nella composizione finale; al contrario, '**cut**' accumula soltanto, senza mostrare il suo contenuto. Inoltre, si osservi che l'elemento vuoto '**paste**' recupera quanto accumulato fino a quel punto; se in seguito, nel corso del documento si usano ancora gli elementi '**cut**' e '**copy**' per accumulare nella stessa area di memoria, questa viene espansa ulteriormente e con un successivo elemento '**paste**' si ottiene tutto, anche quanto già incollato in precedenza.

La copia avviene utilizzando il codice del sistema di composizione finale e ciò ha, come effetto collaterale, il pregio di mantenere inalterata la numerazione degli elementi '**object**', dove i riferimenti automatici, puntano correttamente ai riquadri originali.

Il difetto di questo sistema di copia sta nell'impossibilità di incollare prima ciò che nel documento appare dopo.

Il fatto che gli elementi '**cut**' e '**copy**' possano essere usati indifferentemente in un contesto a blocchi o lineare e che possano contenere indifferentemente questo e quello, implica che il loro utilizzo richieda accortezza. In particolare, non si devono accumulare nella stessa area di memoria dei blocchi assieme a dei componenti lineari; inoltre, l'elemento '**paste**' va usato nel contesto appropriato al contenuto dell'area di memoria che si vuole incollare. Evidentemente, il sistema di controllo SGML non è in condizione di individuare errori di utilizzo di questo tipo.

Copia di piè di pagina

Può essere interessante la copia di una nota a piè di pagina, per poi riprodurla tale e quale in altre posizioni. Si osservi l'esempio seguente:

```
<p>Bla bla bla bla<copy copy="nota"><footnote>Il classico testo di nessuna
importanza</footnote></copy> bla bla bla...</p>
<p>Di nuovo bla bla bla bla<paste paste="nota"> bla bla bla...</p>
```

In questo modo, si intende avere una sola nota a piè di pagina, per entrambe le posizioni: sia la prima volta, dove si vede l'uso dell'elemento '**footnote**', sia dopo, quando viene incollato il contenuto dell'area di memoria '**nota**'. In pratica, la nota che si legge alla fine del capitolo è una sola e i riferimenti alla nota, sono sempre allo stesso numero di nota.

Copia di immagini

È possibile utilizzare gli elementi '**cut**', '**copy**' e '**paste**' anche per accumulare delle immagini che si inseriscono con gli elementi '**∗img**'. Il vantaggio di questo sta nel fatto che nella composizione finale, viene prodotto un solo file contenente l'immagine stessa. Pertanto, ciò può essere molto utile per rappresentare delle icone ricorrenti nel documento.

Riferimenti incrociati e ipertestuali 449
Note e piè pagina 451
Riferimenti esterni e citazioni 451
Indici analitici e termini speciali 452
Esempio di indice analitico 454
Esempio di indice di termini speciali 454
Caratteristiche del software e di altri «lavori» 455
Informazioni su sezioni specifiche del documento 456
Sezioni particolari 457

Alml ha una gestione abbastanza ricca delle informazioni che si ricollegano attraverso riferimenti incrociati. La caratteristica fondamentale di Alml è di far sì che tutto ciò che è fruibile in forma elettronica, rimanga accessibile anche in forma stampata su carta. Per esempio, non è prevista la possibilità di annotare un riferimento ipertestuale a una risorsa di cui non si possa vedere l'indirizzo nella forma stampata.

Riferimenti incrociati e ipertestuali

I riferimenti incrociati si realizzano attraverso l'indicazione di ancore (o etichette se si preferisce il termine) e di puntatori a tali ancore. Esistono diversi modi per definire un'ancora e un riferimento a questa: tutti gli elementi che dispongono di un attributo '**id**', sono ancore oppure sono puntatori alle ancore.

Gli elementi usati per delimitare i titoli dei volumi, delle parti, dei capitoli, delle sezioni e dei riquadri (figure, tabelle, ecc.), sono ancore a cui si può puntare, ma per inserire un'ancora nel testo normale, è possibile usare l'elemento vuoto '**anchor**', anche questo provvisto di attributo '**id**'. Tuttavia, l'elemento '**anchor**' è speciale, perché provvisto anche dell'attributo '**type**', con cui è possibile stabilire se si voglia un'ancora invisibile oppure visibile. L'esempio seguente inserisce un'ancora visibile, mentre se si omette l'attributo '**type**', l'ancora è invisibile in modo predefinito:

```
<p>Bla bla bla, questa <anchor id="oggetto" type="visible">pentola
è fatto di alluminio, bla bla bla...</p>
```

Un'ancora visibile è qualcosa che viene mostrato in modo evidente nella composizione stampata; il suo scopo è quello di poter fare dei riferimenti a posizioni esatte nel testo. Per esempio, la parola «visibile» di questo paragrafo si trova nella posizione u0.1:*a*. Se l'ancora non fosse visibile, il riferimento che si otterrebbe riguarderebbe soltanto la sezione in cui questa è contenuta.

Esistono due elementi vuoti per fare riferimento alle ancore: '**sectionref**', per ottenere un riferimento alla sezione in cui si trova l'ancora e '**objectref**' per fare riferimento a un riquadro. In particolare, l'elemento '**objectref**' può essere usato anche senza l'attributo '**id**' per fare riferimento all'ultima ancora di un riquadro, per semplificare la scrittura delle didascalie.

Quando si realizza un documento che può includere o meno una certa porzione a cui puntano alcuni riferimenti, per evitare che vengano mostrati questi collegamenti mancanti, si può usare l'elemento '**ifref**', con il quale si delimita la parte da non comporre se manca il riferimento indicato nell'attributo '**id**'. D'altro canto, per ottenere l'effetto opposto, di mostrare qualcosa solo se manca un riferimento, si può usare l'elemento '**ifnotref**'.

Tabella u69.2. Elementi utili nella gestione dei riferimenti incrociati.

Elemento	Descrizione
`tomeheading [id="`***ancora***`"] [lang="…"] [bookmark="…"]`	Titolo del volume. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`h0 [id="`***ancora***`"] [lang="…"] [bookmark="…"]`	Titolo della parte. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`h1 [id="`***ancora***`"] [lang="…"] [bookmark="…"]`	Titolo del capitolo. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`h2 [id="…"] [bookmark="…"]`	Titolo della sezione. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`h3 [id="`***ancora***`"] [bookmark="…"]`	Titolo della sottosezione. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`h4 [id="`***ancora***`"] [bookmark="…"]`	Titolo della sotto-sottosezione. L'attributo '**id**' consente di specificare un'ancora di riferimento.
`object [id="`***ancora***`"] [pos="fixed|float"] [sep="none|rule|border"] [split="0|1"] [printedfontsize="`***dimensione***`"]`	Involucro di un riquadro. L'attributo '**id**' consente di mette un'ancora di riferimento.
`anchor id="`***ancora***`" [type="hidden|visible"]`	Elemento vuoto per l'inserimento di un'ancora in un contesto lineare. L'attributo '**id**' consente di specificare l'ancora di riferimento. L'attributo '**type**' consente rendere visibile l'ancora; il valore '**invisible**' è predefinito.
`sectionref id="`***riferimento***`"`	Elemento vuoto per l'inserimento di un riferimento a un'ancora, individuata dal valore assegnato all'attributo '**id**'. Il riferimento individua il numero di una sezione, indicando generalmente il capitolo e, se presenti, le suddivisioni inferiori, oppure la parte, oppure il volume, se il contesto è esterno al capitolo.
`objectref id="`***riferimento***`"`	Elemento vuoto per l'inserimento di un riferimento a un'ancora di un elemento '**object**', individuata dal valore assegnato all'attributo '**id**'.
`ifref id="`***riferimento***`"`	Si tratta di un elemento che può essere inserito in un contesto lineare o tra i blocchi e può contenere sia un testo lineare, sia dei blocchi. Il contenuto dell'elemento viene reso nella composizione tipografica solo se l'ancora indicata con l'attributo '**id**' esiste.

Elemento	Descrizione
`ifnotref id="`***riferimento***`"`	Si tratta di un elemento che può essere inserito in un contesto lineare o tra i blocchi e può contenere sia un testo lineare, sia dei blocchi. Il contenuto dell'elemento viene reso nella composizione tipografica solo se l'ancora indicata con l'attributo '**id**' non esiste.

Note e piè pagina

Alml prevede l'utilizzo di due tipi di annotazioni: avvertimenti che devono risaltare in un riquadro e note a piè pagina. Le note evidenziate sono indicate all'interno di un elemento '**frame**', mentre quelle a piè pagina sono inserite nell'elemento '**footnote**'. Le note a piè pagina vengono inserite nell'elemento '**footnote**', che si colloca all'interno delle righe; al contrario, l'elemento '**frame**' è un blocco che contiene blocchi.

```
<frame>
    <p>Attenzione! Si tratta di un'operazione rischiosa.</p>
</frame>
```

L'esempio precedente mostra l'utilizzo di un riquadro, mentre quello successivo mostra l'uso di un piè pagina.

```
<p>Bla bla bla<footnote>Questa parola si ripete.</footnote> bla bla...</p>
```

Tabella u69.5. Annotazioni a vario titolo.

Elemento	Descrizione
`frame`	Riquadro di avvertimento.
`footnote`	Nota a piè pagina.

Purtroppo, nella composizione stampata, le tabelle che si possono disporre su più pagine interferiscono con le note a piè di pagina. Si tratta di un difetto di LaTeX (precisamente del pacchetto '**longtable**'). Per risolvere il problema, si è reso necessario raggruppare le note alla fine dei capitoli.

Riferimenti esterni e citazioni

Alcuni elementi sono specializzati per fare riferimento a qualcosa di esterno. Il caso più comune riguarda l'elemento '**uri**', con il quale si indica un indirizzo ipertestuale esterno al documento:

```
<p>Bla bla bla <uri><![CDATA[http://www.brot.dg]]></uri> bla bla...</p>
```

Per indicare il riferimento a una pagina di manuale, si può usare l'elemento '**man**', in modo da ottenere una rappresentazione uguale a quella tradizionale, ma ciò non comporta alcun richiamo automatico alla visualizzazione di tale pagina di manuale:

```
<p>Bla bla bla <man>ls<mansect>1</mansect></man> bla bla...</p>
```

Figura u69.8. Sintassi per l'uso dell'elemento '**man**'.

```
man
 |--nome
 '--mansect
      '--n_sezione
```

La tabella u69.9 riepiloga questi e altri elementi affini.

Tabella u69.9. Riferimenti esterni.

Elemento	Descrizione
`uri`	Si tratta di un elemento che si inserisce in un contesto letterale e contiene il riferimento a un URI esterno, da indicare all'interno di una sezione marcata di tipo CDATA.
`uristr`	Si utilizza come l'elemento '**`uri`**', con la differenza che il contenuto non viene inserito in una sezione marcata e non si crea alcun riferimento ipertestuale.
`blockquote`	Si tratta di un blocco che contiene una citazione, contenuta in altri blocchi. Alla fine, prima delle conclusione dell'elemento, può apparire l'elemento '**`quoteinfo`**'.
`quoteinfo`	So tratta di un blocco contenente componenti lineari, che serve a fornire informazioni sulla citazione.
`bibref`	Si inserisce in un contesto lineare e contiene componenti lineari. Precisamente si usa per delimitare il titolo di un documento.
`man`	Si inserisce in un contesto lineare e contiene componenti lineari, oltre che l'elemento '**`mansect`**'. Si usa indicare il nome di una pagina di manuale.
`mansect`	Si inserisce all'interno dell'elemento '**`man`**' e contiene un numero, che rappresenta il numero di una sezione della pagina di manuale.

L'elemento '**`uristr`**' è una variante di '**`uri`**', con lo scopo di non generare un riferimento ipertestuale. Ciò può servire per rappresentare un indirizzo di fantasia, oppure un indirizzo reale che non è più valido. Si possono indicare in questo modo anche i nomi a dominio.

L'elemento '**`blockquote`**' è previsto per delimitare una citazione in uno o più blocchi. Alla fine dell'elemento '**`blockquote`**' è prevista la possibilità di usare un solo elemento '**`quoteinfo`**', con lo scopo di contenere informazioni relative alla citazione:

```
<blockquote>
    blocchi
    …
    [<quoteinfo>componenti_lineari…</quoteinfo>]
</blockquote>
```

Figura u69.10. Sintassi per l'uso dell'elemento '**`blockquote`**'.

```
blockquote
 |--blocco…
 '--[<quoteinfo>]
      '--componenti_lineari…
```

Indici analitici e termini speciali

Diversi tipi di elementi nella struttura di Alml sono predisposti per accumulare informazioni da restituire a richiesta. La situazione più semplice è data dalla gestione degli indici analitici, dove con l'elemento '**`indexentry`**' si inserisce una voce nell'indice analitico generale o in un altro individuato da un nome libero:


```
<h1>
I colori dell'arcobaleno
<indexentry>arcobaleno</indexentry>
<indexentry><code>color</code></indexentry>
</h1>
```

L'elemento '**`indexentry`**' appartiene al gruppo di quelli che possono essere inseriti all'interno di una riga; nell'esempio si vede la situazione tipica in cui lo si inserisce nel testo di un titolo. In questo caso, sono state indicate due voci dell'indice analitico generale: la parola «arcobaleno» viene inserita in modo normale, mentre la parola «color» viene inserita con un carattere dattilografico.

Ogni indice analitico ha un nome e quello generale, o predefinito, corrisponde a '**`main`**'. L'esempio mostrato sopra sarebbe perfettamente equivalente a quello seguente:

```
<h1>
I colori dell'arcobaleno
<indexentry index="main">arcobaleno</indexentry>
<indexentry index="main"><code>color</code></indexentry>
</h1>
```

Per recuperare l'elenco di un indice analitico si utilizza l'elemento '**`printindex`**', in cui, lo stesso attributo '**`index`**' permette di stabilire quale indice estrapolare.

Figura u69.13. Sintassi per l'uso dell'elemento '**`indexentry`**'.

```
indexentry [index="nome_indice"]
  '--{testo|code|asciicode|kbd|vkbd|kp|strdfn}…
```

Tabella u69.14. Gestione degli indici analitici.

Elemento	Descrizione
`indexentry [index="nome_indice"]`	Dichiara una voce per l'indice analitico. L'attributo '**`index`**' consente di inserire la voce in un indice analitico particolare; se si omette, si fa riferimento all'indice '**`main`**'.
`special special="nome_indice"`	Delimita un termine speciale, che per qualche ragione si vuole seguire e controllare in un indice analitico specializzato. L'attributo '**`special`**' serve a specificare in quale indice analitico inserire la voce.
`printindex [index="nome_indice"] [indexcontext="all|tome|part↩ ↪ |chapter" ] [indexref="default|section"]`	Si tratta di un elemento vuoto, da usare tra i blocchi, per inserire l'indice analitico accumulato alla voce specificata con l'attributo '**`index`**'. Se si omette l'attributo '**`index`**', si fa riferimento all'indice '**`main`**'. L'attributo '**`indexcontext`**' specifica il contesto a cui si deve riferire l'indice analitico; è predefinito il contesto '**`all`**', che richiede l'indice completo. L'attributo '**`indexref`**' serve a specificare in che modo devono apparire i riferimenti alle voci dell'indice; con la parola chiave '**`section`**', si richiede espressamente che il riferimento sia solo al numero della sezione.

Esiste anche un altro elemento che inserisce voci negli indici analitici; si tratta di '**`special`**', che inserisce una voce nell'indice corrispondente al nome indicato con l'attributo che ha lo stesso nome:

'**special**'.

La differenza tra '**special**' e '**indexentry**' sta nella destinazione, in quanto il primo dovrebbe servire per tracciare l'uso di certi termini e, attraverso l'indice analitico relativo, verificare l'utilizzo uniforme degli stessi. Da un punto di vista puramente operativo, l'elemento '**special**' si distingue da '**indexentry**' perché mostra nella composizione finale il termine che contiene, mentre '**indexentry**' lo nasconde.

Esempio di indice analitico

Viene proposto qui un esempio completo di accumulo di voci in un indice analitico e di riproduzione dell'indice stesso. Vengono usati in particolare tutti gli elementi che possono essere inseriti nelle voci dell'indice, in modo da poterne osservare l'effetto nella riproduzione delle stesse. Si osservi che le voci vengono accumulate nell'indice predefinito '**main**'.

```
<p>Alcuni segnali possono essere inviati al programma con il quale si interagisce attraverso delle
combinazioni di tasti. Di solito si invia un segnale
<indexentry><code>SIGINT</code></indexentry><samp>SIGINT</samp> attraverso il carattere
<indexentry><asciicode>^c</asciicode></indexentry><asciicode>^c</asciicode>, ovvero
<indexentry><asciicode>ETX</asciicode></indexentry><asciicode>ETX</asciicode>, che si ottiene con
la combinazione virtuale <indexentry><vkbd>Control_c</vkbd></indexentry><vkbd>Control_c</vkbd>, a
cui spesso, fortunatamente, corrisponde la combinazione reale
<indexentry><kbd>Ctrl c</kbd></indexentry><kbd>Ctrl c</kbd>.</p>

<p>I pulsanti grafici <button>Next</button> e <button>Prev</button> permettono di passare alla
modalità grafica successiva (quella che si otterrebbe con la combinazione <indexentry>X:
<kbd>Ctrl Alt <kp>+</kp></kbd></indexentry><kbd>Ctrl Alt <kp>+</kp></kbd>) e
precedente (<indexentry>X:
<kbd>Ctrl Alt <kp>-</kp></kbd></indexentry><kbd>Ctrl Alt <kp>-</kp></kbd>).</p>

<printindex>
```

Nel riquadro successivo si vede come può risultare l'esempio nella composizione finale; si osservi che, nella composizione per la stampa, i riferimenti alle pagine potrebbero risultare sfasati, nel caso il riquadro dovesse passare nella pagina successiva:

Alcuni segnali possono essere inviati al programma con il quale si interagisce attraverso delle combinazioni di tasti. Di solito si invia un segnale '**SIGINT**' attraverso il carattere <^c>, ovvero <*ETX*>, che si ottiene con la combinazione virtuale <*Control_c*>, a cui spesso, fortunatamente, corrisponde la combinazione reale [*Ctrl c*].
I pulsanti grafici NEXT e PREV permettono di passare alla modalità grafica successiva (quella che si otterrebbe con la combinazione [*Ctrl Alt* ⊞]) e precedente ([*Ctrl Alt* ⊟]).
<*Control_c*> 454
[*Ctrl c*] 454
<*ETX*> 454
SIGINT 454
X: [*Ctrl Alt* ⊞] 454
X: [*Ctrl Alt* ⊟] 454
<^c> 454

Esempio di indice di termini speciali

Viene proposto un esempio completo per l'utilizzo di '**special**', allo scopo di tenere traccia dell'uso di alcuni nomi.

```
<p>I formati più comuni per la stampa sono <special
special="nome-formato">DVI</special>, <special
special="nome-formato">PostScript</special>, <special
special="nome-formato">PDF</special>. Tra questi, quello che si presta
alle rielaborazioni, per esempio per favorire la rilegatura, è il
formato <special special="nome-formato">PostScript</special>. Per la
consultazione di un documento in modo interattivo, i formati comuni sono
<special special="nome-formato">HTML</special>, <special
special="nome-formato">XHTML</special> e ancora <special
special="nome-formato">PDF</special>.</p>

<p>In questo documento abbiamo parlato di:</p>

<printindex index="nome-formato">
```

Nel riquadro successivo si vede come può risultare l'esempio nella composizione finale; si osservi che, nella composizione per la stampa, i riferimenti alle pagine potrebbero risultare sfasati, a causa del passaggio del riquadro a una pagina successiva:

I formati più comuni per la stampa sono DVI, PostScript, PDF. Tra questi, quello che si presta alle rielaborazioni, per esempio per favorire la rilegatura, è il formato PostScript. Per la consultazione di un documento in modo interattivo, i formati comuni sono HTML, XHTML e ancora PDF.
In questo documento abbiamo parlato di:
DVI 454
HTML 454
PDF 454 454
PostScript 454 454
XHTML 454

Caratteristiche del software e di altri «lavori»

La struttura di Alml dispone di un elemento speciale che si può inserire nel testo lineare, il cui scopo è quello di annotare alcune informazioni sul software e su lavori simili. Si osservi l'esempio seguente:

```
<p>Stiamo parlando di Mpage,<workinfo>
<workname>Mpage</workname>
<worklicense>licenza speciale che non ammette le modifiche</worklicense>
<worklicensetext>

    <p>Permission is granted to anyone to make or distribute verbatim
    copies of this document as received, in any medium, provided that
    this copyright notice is preserved, and that the distributor grants
    the recipient permission for further redistribution as permitted by
    this notice.</p>

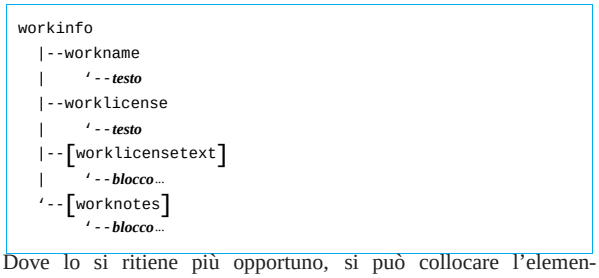
</worklicensetext>
</workinfo> un programma che si occupa di...</p>
```

Nel riquadro successivo si può vedere cosa succede nella composizione finale:

Stiamo parlando di Mpage,[1] un programma che si occupa di...

Solo gli elementi '**workname**' e '**worklicense**' sono obbligatori, dal momento che il loro contenuto appare in un piè pagina locale. L'elemento '**worklicensetext**' è facoltativo e può essere utile per annotare una licenza unica, per la quale non possa essere individuato un riferimento standard; inoltre, un altro elemento, '**worknotes**', permette di annotare qualcosa al riguardo.

Figura u69.21. Sintassi per l'uso dell'elemento '**workinfo**'.



Dove lo si ritiene più opportuno, si può collocare l'elemento '**printworkinfo**', per ottenere l'elenco ordinato di queste informazioni accumulate.

Tabella u69.22. Tracciamento di informazioni sul software citato.

Elemento	Descrizione
workinfo	Contenitore delle informazioni, da inserire in un contesto lineare.
workname	Contiene il nome del software o di altro lavoro.
worklicense	Contiene la denominazione o la descrizione breve della licenza.
worklicensetext	Si usa per riportare il testo della licenza, organizzato in blocchi.
worknotes	Si usa per riportare delle annotazioni, organizzato in blocchi.

Elemento	Descrizione
`printworkinfo [workinforef="default|section"]`	Inserisce le informazioni accumulate in modo ordinato. L'attributo '**`workinforef`**', se riceve il valore '**`section`**', fa sì che i riferimenti ai lavori vengano fatti sempre solo utilizzando i numeri di sezione.

Nel riquadro successivo appare ciò che si potrebbe vedere nella composizione finale, quando si inserisce l'elemento '**`printworkinfo`**':

Mpage, u0.5

Permission is granted to anyone to make or distribute verbatim copies of this document as received, in any medium, provided that this copyright notice is preserved, and that the distributor grants the recipient permission for further redistribution as permitted by this notice.

Informazioni su sezioni specifiche del documento

In situazioni particolari, potrebbe essere necessario, o anche solo utile, tenere traccia dell'origine di una sezione del documento, assieme a delle annotazioni a vario titolo. Per questo si può utilizzare l'elemento '**`docinfo`**', che costituisce un blocco, contenente blocchi. Si osservi l'esempio seguente:

```
<docinfo docinfo="modifiche">
    <dl>
    <dt>2002.09.15</dt>
    <dd>
        <p>Il testo viene aggiornato nel contenuto, con l'inserimento
        della sezione «bla bla bla», da parte di Caio Cai
        (caio@brot.dg).</p>
    </dd>
    <dt>2002.09.08</dt>
    <dd>
        <p>Il testo viene modificato per adeguarlo alla nuova veste
        grafica dell'opera, per opera di Caio Cai (caio@brot.dg);
        il contenuto rimane invariato.</p>
    </dd>
    <dt>2002.02.02</dt>
    <dd>
        <p>Il testo originale è di Tizio Tizi e risale al 2002.02.02.
        Nello stesso giorno, il testo ha subito qualche aggiustamento
        per opera di Caio Cai (caio@brot.dg), con il consenso
        dell'autore.</p>
    </dd>
    </dl>
</docinfo>
```

L'esempio mostra in particolare l'uso dell'elemento '**`docinfo`**' per annotare lo storico delle modifiche fatte su quella porzione di documento; come si può vedere, vengono indicate prima le azioni più recenti, ma questo dipende solo da una scelta organizzativa.

Per ottenere l'elenco delle informazioni accumulate in questo modo, si utilizza l'elemento vuoto '**`printdocinfo`**'. Per inserire l'elenco dell'esempio precedente, va usato così:

```
…
<printdocinfo docinfo="modifiche">
…
```

Nel riquadro seguente si vede ciò che potrebbe apparire nella composizione finale:

sezione u0.6, *Informazioni su sezioni specifiche del documento* , pag. 456

2002.09.15

Il testo viene aggiornato nel contenuto, con l'inserimento della sezione «bla bla bla», da parte di Caio Cai (caio@brot.dg).

2002.09.08

Il testo viene modificato per adeguarlo alla nuova veste grafica dell'opera, per opera di Caio Cai (caio@brot.dg); il contenuto rimane invariato.

2002.02.02

Il testo originale è di Tizio Tizi e risale al 2002.02.02. Nello stesso giorno, il testo ha subito qualche aggiustamento per opera di Caio Cai (caio@brot.dg), con il consenso dell'autore.

Tabella u69.27. Tracciamento di informazioni su sezioni particolari del documento globale.

Elemento	Descrizione
`docinfo [docinfo="`***`nome_gruppo`***`"]`	Blocco contenente blocchi per l'annotazione di qualcosa sul documento. L'attributo '**`docinfo`**' consente di stabilire un raggruppamento a cui appartiene l'informazione accumulata; se non viene fornito, il valore predefinito per l'attributo è '**`default`**'.
`printdocinfo [docinfo="`***`nome_gruppo`***`"]`	Elemento vuoto che si inserisce tra i blocchi, per ottenere l'elenco delle annotazioni associate al nome che si assegna all'attributo '**`docinfo`**'. Se l'attributo non viene fornito, il valore predefinito per l'attributo è '**`default`**'.

Sezioni particolari

È disponibile l'elemento vuoto '**`sectiongroup`**' per inserire il numero della sezione in cui si trova in un elenco particolare, che successivamente può essere ottenuto con l'elemento vuoto '**`printsectiongroup`**'. Nell'esempio successivo, viene annotato che la sezione appartiene al gruppo '**`non-modificabile`**':

```
…
<sectiongroup group="non-modificabile">
…
```

Nell'esempio successivo, si vuole ottenere l'elenco di tutte le sezioni associate al gruppo '**`non-modificabile`**':

```
…
<p>Segue l'elenco delle sezioni dell'opera che non possono
essere modificate, per vari motivi:<p>
<printsectiongroup group="non-modificabile">
…
```

Nel riquadro successivo, si vede ciò che potrebbe apparire nell'elenco:

Segue l'elenco delle sezioni dell'opera che non possono essere modificate, per vari motivi:

appendix A, *GNU GENERAL PUBLIC LICENSE* , pag. 2

appendix B, *GNU Free Documentation License* , pag. 12

Tabella u69.31. Sezioni particolari.

Elemento	Descrizione
`sectiongroup [group="`***gruppo***`"]`	Dichiara che la sezione appartiene al gruppo indicato nell'attributo. Se manca l'attributo, si intende che il gruppo in questione sia denominato '**`nomod`**'.
`printsectiongroup [group="`***gruppo***`"]`	Inserisce l'elenco delle sezioni che appartengono al gruppo indicato nell'attributo. Se manca l'attributo, si intende che il gruppo in questione sia denominato '**`nomod`**'.

[1] **Mpage** licenza speciale che non ammette le modifiche

Immagini e video

Immagini esterne 462
Immagini incorporate Base64 462
Immagini incorporate EPS 463
Immagini incorporate XFig 463
Immagini incorporate LilyPond 464
Immagini incorporate TeX e LaTeX 464
Immagini incorporate Gnuplot 465
Osservazioni sull'incorporazione di codice estraneo 466

Alml consente di inserire immagini provenienti da file esterni, oppure incorporando del codice estraneo, con cui queste devono essere disegnate. Eventualmente si può anche fare riferimento a dei video, ma in tal caso non si ha una «incorporazione» vera e propria nel testo.

Gli elementi con cui si possono incorporare delle immagini o dei video vanno usati in un contesto lineare; pertanto, per poter essere usati in un riquadro ('**`object`**'), come se fossero dei blocchi, vanno inserite dentro l'elemento '**`imgblock`**'.

Inizialmente, il modo più semplice per inserire un'immagine è quello di preparare un file in un formato comune a matrice di punti (come può esserlo un formato PNG) e di collocarlo in una sottodirectory rispetto alla posizione in cui si trova il file sorgente SGML. Per esempio, disponendo del file '`cielo-azzurro.png`' collocato nella directory '`figure/`'; l'immagine si potrebbe incorporare in un testo nel modo seguente:

```
<p>Mi piace vedere un cielo azzurro come in questa piccola icona: <img
imgfile="figure/cielo-azzurro" height="2cm">. Come vorrei stendermi su
un bel prato ad ammirare tanta bellezza.</p>
```

Se invece quello che si vuole è mostrare la figura nel riquadro di un elemento '**`object`**', occorre aggiungere l'elemento '**`imgblock`**':

```
<object>
<caption>
    Figura <objectref>. Il cielo azzurro che vorrei ammirare
    stendendomi su un bel prato...
</caption>
<imgblock>
<img imgfile="figure/cielo-azzurro" width="100%">
</imgblock>
</object>
```

Tutti gli elementi che vengono descritti in questo capitolo per l'inserzione delle immagini, vanno usati in un contesto lineare, oppure, vanno inserite in un elemento '**`imgblock`**' per poter apparire come blocchi in un elemento '**`object`**'.

Tabella u70.3. Elementi SGML che servono a incorporare delle immagini.

Elemento	Descrizione
`img imgfile="`***file***`" [alt="`***descrizione***`"] [height="`***altezza***`"] [width="`***larghezza***`"]`	Elemento vuoto per incorporare un file esterno, indicato nell'attributo '`imgfile`', ma senza estensione. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.

Elemento	Descrizione
`embimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine trasformata con l'algoritmo Base64. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`epsimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato EPS letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`figimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato XFig letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`lyimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato LilyPond letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.

Elemento	Descrizione
`teximg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato TeX letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`lateximg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato LaTeX letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`gnuplotimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato Gnuplot letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`eukleidesimg` [`alt="`***descrizione***`"`] [`file="`***file***`"`] [`height="`***altezza***`"`] [`width="`***larghezza***`"`]	Elemento per incorporare un'immagine in formato Eukleides letterale. L'attributo '**`alt`**' consente di specificare una descrizione alternativa nel caso non si potesse visualizzare l'immagine; l'attributo '**`file`**' consente di salvare una copia del contenuto dell'elemento nel file indicato; gli attributi '**`width`**' e '**`height`**' consentono di specificare le dimensioni dell'immagine.
`video` [`src="`***file***`"`] [`title="`***titolo***`"`] [`artist="`***artista***`"`] [`copyright="`***condizioni***`"`]	Elemento per fare riferimento a un video, rappresentato dal percorso da indicare con l'attributo '**`src`**'. Il video viene convertito in OGV, ma soltanto nella composizione HTML; tuttavia, se poi segue una composizione PDF, i collegamenti esterni vengono fatti corrispondere agli stessi file OGV.

Elemento	Descrizione
`imgblock`	Elemento che può contenere uno o più elementi '**`*img`**' da inserire all'interno in un elemento '**`object`**'.

Dalla tabella appena mostrata si può osservare che esistono degli attributi comuni; in modo particolare '**`height`**' e '**`width`**'. L'attributo '**`height`**' serve a specificare l'altezza dell'immagine, mentre l'attributo '**`width`**' specifica la larghezza. Se vengono forniti entrambi, l'immagine ottenuta dalla composizione rispetta entrambi i valori, pertanto può risultare deformata; se invece si specifica uno solo dei due valori, l'altro viene determinato in proporzione. Le misure vanno espresse nei modi riconoscibili da Alml; per esempio si può scrivere '**`height="5cm"`**', oppure '**`height="50mm"`**', oppure '**`height="1.969in"`**',... Quando si tratta della larghezza ('**`width`**'), la misura può essere espressa anche in modo percentuale, riferendosi allo spazio disponibile. Per esempio, '**`width="100%"`**' richiede la larghezza massima in base al contesto. Logicamente, in condizioni normali è preferibile usare sempre solo l'attributo '**`width`**' con un valore percentuale.

È possibile evitare l'uso di entrambi gli attributi '**`width`**' e '**`height`**', quando l'immagine contiene delle misure proprie; in tal caso, le misure originali vengono rispettate.

Quando si genera una composizione in formato HTML, le misure devono essere trasformate in punti grafici (*pixel*). Considerato che mediamente uno schermo grafico viene usato alla risoluzione di 1024×768, per ogni punto tipografico PostScript (ovvero punti da 1/72 in) si ottengono due punti grafici.

Un altro attributo comune a tutti gli elementi che inseriscono un'immagine è '**`alt`**', che serve a descrivere brevemente l'immagine. Questa informazione serve nella composizione HTML, per mostrare una descrizione minima in caso di problemi nella visualizzazione dell'immagine.

Quando gli elementi incorporano il codice che rappresenta l'immagine, questo deve essere racchiuso in una sezione marcata di tipo '**`CDATA`**', per non essere alterato in alcun modo; inoltre, per tali elementi è disponibile l'attributo '**`file`**', con il quale è possibile salvare, in fase di composizione, una copia di quel contenuto nel file indicato. Il file in questione viene salvato soltanto se la directory di destinazione esiste già e se, oltre ad avere i permessi necessari, non esiste già un file con quel nome.

Immagini esterne

Nella parte iniziale del capitolo sono già apparsi degli esempi di utilizzo dell'elemento '**`img`**', per l'inserimento di un'immagine proveniente da un file esterno. Come già spiegato, è bene che il file in questione si trovi in una directory differente rispetto a quella in cui si trova il file SGML sorgente.

Il file viene indicato nell'attributo '**`imgfile`**' senza l'estensione, perché vengono tentate automaticamente diverse possibilità, partendo da formati che dovrebbero offrire una qualità maggiore.

Immagini incorporate Base64

Per incorporare un'immagine codificata con l'algoritmo Base64 si può usare il programma Uuencode, oppure Mpack, descritti nella sezione 39.12. Supponendo di utilizzare Uuencode e di volere inserire l'immagine contenuta nel file '`prova.jpg`', basta procedere come segue:

```
$ uuencode -m prova.jpg ciao > prova.uuencode [Invio]
```

Quello che si ottiene in questo caso è il file '`prova.uuencode`', che può apparire simile al testo seguente, che è stato ridotto per comodità:


```
begin-base64 664 ciao
JSFQUy1BZG9iZS0yLjAKJSVDcmVhdG9yOiAiYmFyY29kZSIsIGxpYmJhcmNv
ZGUgc2FtcGxlIGZyb250ZW5kCiUgJSVEb2N1bWVudFBhcGVyU2l6ZXM6IGE0
…
…
b3cKMTA0LjAwIDEwLjAwIG1vdmV0byAoOSkgc2hvdwoKJSBFbmQgYmFyY29k
ZSBmb3IgIjk5MTIzNDU2Nzg5MCIKCiUlRW5kUGFnZQoKc2hvd3BhZ2UKJSVU
cmFpbGVyCiUlRU9GCgo=
====
```

Da questo file, ottenuto con Uuencode, va tolta la prima e l'ultima riga; il resto si può inserire in un elemento '**`embimg`**'. Viene mostrato un esempio:

```
<p>Bla bla bla
<embimg alt="Esempio" width="10%">
<![CDATA[
JSFQUy1BZG9iZS0yLjAKJSVDcmVhdG9yOiAiYmFyY29kZSIsIGxpYmJhcmNv
ZGUgc2FtcGxlIGZyb250ZW5kCiUgJSVEb2N1bWVudFBhcGVyU2l6ZXM6IGE0
…
…
b3cKMTA0LjAwIDEwLjAwIG1vdmV0byAoOSkgc2hvdwoKJSBFbmQgYmFyY29k
ZSBmb3IgIjk5MTIzNDU2Nzg5MCIKCiUlRW5kUGFnZQoKc2hvd3BhZ2UKJSVU
cmFpbGVyCiUlRU9GCgo=
]]>
</embimg> bla bla bla.</p>
```

Dal momento che si vuole evitare qualunque interpretazione SGML, è necessario racchiudere il contenuto di questi elementi in una sezione marcata di tipo CDATA, così come si può vedere nell'esempio appena apparso.

Immagini incorporate EPS

Si può incorporare codice EPS utilizzando l'elemento '**`epsimg`**'. Viene mostrato un esempio:

```
<p>Bla bla bla
<epsimg alt="Esempio" width="10%">
<![CDATA[
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: Daniele Giacomini
%%BoundingBox: 0 0 100 100
%%EndComments
50 50 translate
gsave
% Cambia e ruota il piano cartesiano
36  {10 rotate 10 10 moveto 30 30 lineto} repeat
stroke
grestore
showpage
%%Trailer
%%EOF
]]>
</epsimg> bla bla bla.</p>
```

Nel riquadro successivo si vede il risultato della composizione:

Bla bla bla  bla bla bla.

Durante la fase di composizione, l'immagine viene trasformata in modo appropriato con degli strumenti, che a volte si limitano a considerare, nel codice originario, l'area di un foglio in formato Lettera verticale (8,5 in × 11 in). In questo caso, ciò che esce dai margini del formato Lettera può risultare escluso. Di tale limite è necessario essere consapevoli quando si preparano immagini del genere.

Immagini incorporate XFig

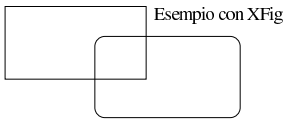
Si può incorporare codice XFig utilizzando l'elemento '**`figimg`**'. Viene mostrato un esempio:

```
<object id="f-esempio-1">
<caption>
    Figura <objectref>. Esempio con <special special="name">XFig</special>.
</caption>
<imgblock>
<figimg alt="Esempio" width="50%">
<![CDATA[
#FIG 3.2
Portrait
Center
Metric
A4
100.00
Single
```

```
-2
1200 2
2 2 0 1 0 7 50 0 -1 0.000 0 0 -1 0 0 5
         270 225 1755 225 1755 990 270 990 270 225
2 4 0 1 0 17 50 0 -1 0.000 0 0 7 0 0 5
         2745 1395 2745 540 1215 540 1215 1395 2745 1395
4 0 0 50 0 0 12 0.0000 4 180 1350 1845 360 Esempio con XFig\001
]]>
</figimg>
</imgblock>
</object>
```

Nel riquadro successivo si vede il risultato della composizione:

Figura u70.10. Esempio con XFig.



Si osservi che gli strumenti usati da Alml per l'inclusione di questo formato, leggono dal codice originario soltanto l'area corrispondente a un foglio in formato Lettera verticale (8,5 in × 11 in), ignorando il resto.

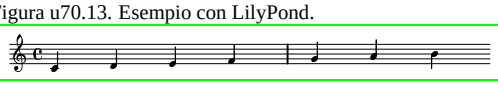
Immagini incorporate LilyPond

L'elemento '**`lyimg`**' consente di incorporare codice LilyPond. Viene mostrato un esempio:

```
<object sep="border" id="f-esempio-2">
<caption>
    Figura <objectref>. Esempio con <special special="name">LilyPond</special>.
</caption>
<imgblock>
<lyimg alt="Esempio" width="40%">
<![CDATA[
\version "2.4.0"
\header {
    tagline = ""
}
\score {
   {c' d' e' f' g' a' b'}
  \layout {
    linewidth = 50
    firstpagenumber = "no"
  }
  \midi {}
}
]]>
</lyimg>
</imgblock>
</object>
```

Nel riquadro seguente, si vede il risultato della composizione dell'esempio; si osservi che l'esempio utilizzava, a sua volta, un riquadro bordato:

Figura u70.13. Esempio con LilyPond.



Nella composizione in formato HTML, in corrispondenza dell'immagine che riproduce il codice musicale di LilyPond, se previsto, si raggiunge il file MIDI corrispondente come riferimento ipertestuale. In pratica, di solito si ottiene di eseguire il brano visualizzato, facendo un clic sull'immagine.

Immagini incorporate TeX e LaTeX

Sono disponibili gli elementi '**`teximg`**' e '**`lateximg`**' per inserire direttamente il codice TeX e LaTeX nel sorgente. Per la precisione, nel caso di '**`teximg`**' vengono aggiunte automaticamente all'inizio due istruzioni, '**`\nonstopmode`**' e '**`\nopagenumbers`**', inoltre, alla fine viene aggiunta l'istruzione '**`\bye`**'; invece, nel caso di '**`lateximg`**' viene aggiunta l'istruzione '**`\nonstopmode`**' all'inizio e '**`\end{document}`**' alla fine.

Il codice LaTeX che viene inserito deve includere tutto il necessario a funzionare correttamente, ma l'aggiunta dell'istruzione '**`\end{document}`**' in modo automatico non può far male se questa è già stata inserita correttamente.

Segue un esempio riferito all'inclusione di codice TeX, dove si può osservare che non viene specificata la dimensione dell'immagine:

```
<object id="f-esempio-3">
<caption>
    Figura <objectref>. Ecco una bella formula.
</caption>
<imgblock>
<teximg alt="Esempio">
<![CDATA[
$$ \chi^2 = \sum_{i=1}^N
          \left (y_i - (a + b x_i)
          \over \sigma_i\right)^2 $$
]]>
</teximg>
</imgblock>
</object>
```

Il riquadro seguente mostra il risultato della composizione:

Figura u70.16. Ecco una bella formula.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (a + bx_i)}{\sigma_i} \right)^2$$

Segue un esempio simile, realizzato con l'inclusione di codice LaTeX; si osservi in particolare la necessità di definire il tipo di documento e il tipo di pagina più semplice previsto. Questa volta si vuole attribuire una dimensione orizzontale all'immagine:

```
<object id="f-esempio-4">
<caption>
    Figura <objectref>. Ecco un'altra bella formula.
</caption>
<imgblock>
<lateximg alt="Esempio" width="100%">
<![CDATA[
\documentclass{article}
\pagestyle{empty}
\begin{document}
$$ \chi^2 = \sum_{i=1}^N
          \left (y_i - (a + b x_i)
          \over \sigma_i\right)^2 $$
\end{document}
]]>
</lateximg>
</imgblock>
</object>
```

Il riquadro seguente mostra il risultato della composizione:

Figura u70.19. Ecco un'altra bella formula.

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (a + bx_i)}{\sigma_i} \right)^2$$

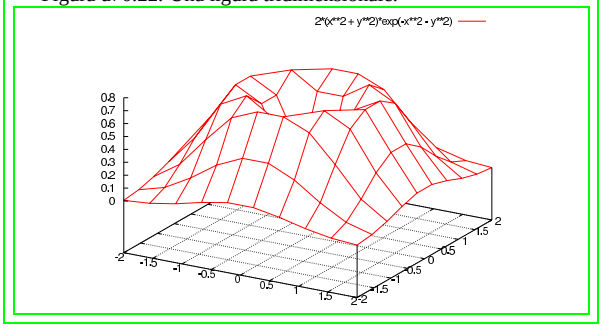
Immagini incorporate Gnuplot

È possibile incorporare codice Gnuplot attraverso l'elemento '**`gnuplotimg`**', che si usa come gli altri elementi simili. In questo caso, viene aggiunta automaticamente l'istruzione '**`set terminal postscript eps color`**' all'inizio. Segue un esempio:

```
<object sep="border">
<caption>
    Figura <objectref>. Una figura tridimensionale.
</caption>
<gnuplotimg alt="2*(x**2 + y**2)*exp(-x**2 - y**2)" width="75%">
<![CDATA[
set grid
set hidden3d
splot [-2:2] [-2:2] 2*(x**2 + y**2)*exp(-x**2 - y**2)
]]>
</gnuplotimg>
</object>
```

Nel riquadro successivo, appare il risultato della composizione; si osservi che l'esempio utilizzava, a sua volta, un riquadro bordato:

Figura u70.22. Una figura tridimensionale.



Osservazioni sull'incorporazione di codice estraneo

Quando si va a incorporare codice esterno, come si fa per esempio con gli elementi '**lateximg**', '**figimg**', '**lyimg**',... è importante evitare di lasciare il codice ASCII *<HT>*. In pratica, una volta inserito il codice nel sorgente SGML, conviene «espandere» il sorgente stesso in modo che anche i caratteri di tabulazione siano trasformati in spazi normali. L'esempio seguente dovrebbe essere sufficientemente chiaro così:

```
$ expand < prima.sgml > dopo.sgml [Invio]
```

Un altro aspetto da considerare è la codifica: se questo codice che si inserisce contiene caratteri che corrispondono a punti di codifica oltre U+007F, ovvero oltre la codifica ASCII pura e semplice, si possono creare dei problemi, che consistono nel non ottenere esattamente gli stessi caratteri di partenza.

Alml consente di realizzare delle tabelle attraverso l'elemento '**tabular**', che deve trovarsi necessariamente all'interno di un elemento '**object**'. Una tabella potrebbe essere realizzata disegnando una figura e incorporandone il codice attraverso uno dei tanti elementi '***img**', ma la tabella ottenuta attraverso l'elemento '**tabular**', tra le altre cose, ha il vantaggio di poter essere divisa tra le pagine nella composizione destinata alla stampa.

Segue un esempio molto semplice di tabella realizzata con l'elemento '**tabular**':

```
<object id="t-esempio-1">
<caption>
    Tabella <objectref>. Ecco il mio primo esempio.
</caption>
<tabular col="2">
<thead>
<trow> Dispositivo  <colsep> Descrizione              </trow>
</thead>
<tbody>
<trow> /dev/fd0     <colsep> Prima unità a dischetti. </trow>
<trow> /dev/hda     <colsep> Primo disco fisso ATA.   </trow>
<trow> /dev/hdb     <colsep> Secondo disco fisso ATA. </trow>
<trow> /dev/sda     <colsep> Primo disco SCSI.        </trow>
<trow> /dev/lp0     <colsep> Prima porta parallela.   </trow>
<trow> /dev/ttyS0   <colsep> Prima porta seriale.     </trow>
</tbody>
</tabular>
</object>
```

Nel riquadro successivo si vede il risultato nella composizione finale:

Tabella u71.3. Ecco il mio primo esempio.

Dispositivo	Descrizione
/dev/fd0	Prima unità a dischetti.
/dev/hda	Primo disco fisso ATA.
/dev/hdb	Secondo disco fisso ATA.
/dev/sda	Primo disco SCSI.
/dev/lp0	Prima porta parallela.
/dev/ttyS0	Prima porta seriale.

L'esempio mostrato è sufficientemente completo: l'elemento '**tabular**' ha un attributo obbligatorio, '**col**', con il quale è necessario dichiarare subito la quantità di colonne che compone la tabella. Le righe della tabella sono raggruppate in due gruppi: l'intestazione, delimitata dall'elemento '**thead**', e il corpo, delimitato dall'elemento '**tbody**'. Le righe sono definite dall'elemento '**trow**' e la separazione tra una colonna e l'altra avviene con l'elemento vuoto '**colsep**'.

Figura u71.4. Sintassi semplificata per l'uso dell'elemento '**tabular**'.

```
tabular col="n_colonne" [columnfractions="suddivisione"]
  |     [printedfontsize="dimensione"] [border="0|1"]
  |--[thead]
  |    '--trow…
  |         |--contenuto_cella
  |         '--[colsep [contenuto_cella]]…
  '--tbody
       '--trow…
            |--contenuto_cella
            '--[colsep [contenuto_cella]]…
```

Tabella u71.5. Elementi SGML che servono a rappresentare le tabelle standard di Alml

Elemento	Descrizione
`tabular col="`*`n_colonne`*`" [columnfractions="`*`suddivisione`*`"] [printedfontsize="`*`dimensione`*`"] [border="0|1"]`	Dichiarazione della tabella. L'attributo '**`col`**' indica la quantità di colonne; l'attributo '**`columnfractions`**' descrive la larghezza delle colonne in proporzione allo spazio orizzontale disponibile; l'attributo '**`printedfontsize`**' consente di dichiarare la dimensione del carattere standard del testo contenuto nelle celle; l'attributo '**`border`**' consente di avere una bordatura più o meno ricca.
`thead`	Contiene le righe di intestazione.
`tbody`	Contiene le righe del corpo.
`trow`	Contiene le celle di una riga.
`colsep`	È un elemento vuoto che separa le colonne delle righe.

L'uso dell'attributo '**`columnfractions`**' potrebbe essere poco intuitivo: una volta dichiarato con l'attributo '**`col`**' la quantità di colonne esistenti, all'attributo '**`columnfractions`**' si assegna una stringa contenente un elenco di valori inferiori a uno, che rappresentano la percentuale di larghezza che deve avere ogni colonna. Per esempio, '**`col="2" columnfractions="0.75 0.25"`**' indica che si tratta di due colonne, dove la prima occupa il 75 % dello spazio orizzontale e la seconda ne occupa il 25 %. In generale, se si usa l'attributo '**`columnfractions`**' conviene che la somma dei valori percentuali dia esattamente il 100 % (pari semplicemente a uno), ma volendo, si può ottenere anche un valore inferiore, per ottenere una tabella che occupa meno spazio orizzontale. Si osservi che se non si usa l'attributo '**`columnfractions`**', il contenuto delle celle può essere esclusivamente di tipo lineare (niente blocchi) e la larghezza delle colonne si estende per tutto lo spazio necessario a contenere il testo senza andare a capo.

L'esempio seguente mostra il caso di una tabella in cui le celle possono contenere più di una riga:

```
<object id="t-tex-controllo-paragrafo-comune">
<caption>
    Tabella <objectref>. Esempio di tabella un po' più complessa.
</caption>
<tabular col="3" columnfractions="0.2 0.4 0.4" border="1">
<thead>
  <trow>Parola di controllo
<colsep>Competenza
<colsep>Condizione o valore predefinito
</trow>
</thead>
<tbody>
  <trow><syntax>\hoffset</syntax>
<colsep>Posizione iniziale dei paragrafi nella pagina.
<colsep><num>0</num>
</trow>
  <trow><syntax>\hsize</syntax>
<colsep>Larghezza del paragrafo a partire da <samp>\hoffset</samp>.
<colsep><num>6,5</num> pollici
</trow>
  <trow><syntax>\parindent</syntax>
<colsep>Rientro della prima riga.
<colsep><num>20</num> punti
</trow>
  <trow><syntax>\baselineskip</syntax>
<colsep>Distanza tra la base di una riga e la base della riga successiva.
<colsep><num>12</num> punti
</trow>
  <trow><syntax>\parskip</syntax>
<colsep>Distanza aggiuntiva tra i paragrafi.
<colsep><num>0</num>
</trow>
  <trow><syntax>\raggedright</syntax>
<colsep>Allinea il testo a sinistra.
<colsep>allineato simultaneamente a sinistra e a destra
</trow>
  <trow><syntax>\leftskip</syntax>
<colsep>Rientro sinistro complessivo.
<colsep><num>0</num>
</trow>
  <trow><syntax>\rightskip</syntax>
<colsep>Rientro destro complessivo.
<colsep><num>0</num>
</trow>
</tbody>
</tabular>
</object>
```


Il riquadro successivo mostra il risultato nella composizione finale:

Tabella u71.8. Esempio di tabella un po' più complessa.

Parola di controllo	Competenza	Condizione o valore predefinito
`\hoffset`	Posizione iniziale dei paragrafi nella pagina.	0
`\hsize`	Larghezza del paragrafo a partire da '**`\hoffset`**'.	6,5 pollici
`\parindent`	Rientro della prima riga.	20 punti
`\baselineskip`	Distanza tra la base di una riga e la base della riga successiva.	12 punti
`\parskip`	Distanza aggiuntiva tra i paragrafi.	0
`\raggedright`	Allinea il testo a sinistra.	allineato simultaneamente a sinistra e a destra
`\leftskip`	Rientro sinistro complessivo.	0
`\rightskip`	Rientro destro complessivo.	0

Come accennato, purché si utilizzi l'attributo '**`columnfractions`**', è possibile inserire nelle celle alcuni elementi che rappresentano blocchi di testo; per esempio: '**`syntax`**', '**`command`**', '**`pre`**' e '**`verbatimpre`**', come in parte si vede nell'esempio già apparso. Ciò dovrebbe consentire l'uso delle tabelle per realizzare degli schemi riassuntivi riferiti a comandi, sintassi o simili. Si osservi l'esempio seguente:

```
<object id="a2-esempio-sintassi-in-tabella">
<tabular col="2" columnfractions="0.618 0.382" border="1">
<thead>
  <trow>Comando
<colsep>Descrizione
</trow>
</thead>
<tbody>
  <trow><syntax>mbadblock <var>unità_dos</var></syntax>
<colsep>Scandisce un'unità &DOS; alla ricerca di settori difettosi.
</trow>
  <trow><syntax>mcd <synsqb><var>directory_dos</var></synsqb></syntax>
<colsep>Permette di modificare o conoscere la directory corrente delle unità &DOS;.
</trow>
  <trow><syntax>mdel <var>file_dos</var><synellipsis></syntax>
<colsep>Cancella i file &DOS; indicati come argomento.
</trow>
  <trow><syntax>mdeltree <var>directory_dos</var><synellipsis></syntax>
<colsep>Cancella le directory &DOS; indicate come argomento.
</trow>
  <trow><syntax>mmd <var>directory_dos</var><synellipsis></syntax>
<colsep>Crea le directory &DOS; indicate come argomento.
</trow>
  <trow><syntax>mmove <var>origine_dos</var><synellipsis> <var>destinazione_dos</var></syntax>
<colsep>Sposta o rinomina uno o più file e directory.
</trow>
  <trow><syntax>mrd <var>directory_dos</var><synellipsis></syntax>
<colsep>Elimina le directory indicate come argomento, purché siano vuote.
</trow>
  <trow><syntax>mren <var>origine_dos</var><synellipsis> <var>destinazione_dos</var></syntax>
<colsep>Rinomina o sposta uno o più file e directory.
</trow>
</tbody>
</tabular>
</object>
```

Il riquadro successivo mostra il risultato nella composizione finale:

Comando	Descrizione
`mbadblock` ***unità_dos***	Scandisce un'unità Dos alla ricerca di settori difettosi.
`mcd` [***directory_dos***]	Permette di modificare o conoscere la directory corrente delle unità Dos.
`mdel` ***file_dos***…	Cancella i file Dos indicati come argomento.
`mdeltree` ***directory_dos***…	Cancella le directory Dos indicate come argomento.
`mmd` ***directory_dos***…	Crea le directory Dos indicate come argomento.
`mmove` ***origine_dos***… ***destinazione_dos***	Sposta o rinomina uno o più file e directory.
`mrd` ***directory_dos***…	Elimina le directory indicate come argomento, purché siano vuote.
`mren` ***origine_dos***… ***destinazione_dos***	Rinomina o sposta uno o più file e directory.

La scelta del rapporto tra le due colonne della tabella, 61,8 % e 38,2 %, rappresenta quello che è noto come «rapporto aureo». Volendo seguire la stessa logica per una tabella di tre colonne, i rapporti sono: 19,1 %, 30,1 % e 50,0 %.

Le tabella molto lunghe possono essere realizzate in modo da consentire il salto pagina, utilizzando l'attributo '**`split`**' nell'elemento '**`object`**' che le contiene. In ogni caso, perché ci possa essere una tabella suddivisibile tra le pagine, è necessario che questa non sia fluttuante.

Il corpo del carattere «normale» che si inserisce all'interno delle celle di una tabella ottenuta con l'elemento '**`tabular`**', può essere controllato nell'intestazione con un elemento '**`printedfontsize`**', come nell'esempio seguente:

```
<head>
    <admin>
        …
        <printedfontsize type="table">3,5mm</printedfontsize>
        …
    </admin>
    …
</head>
```

Se non si indica questa informazione, né nell'intestazione, né nell'elemento '**`tabular`**', il carattere viene comunque ridotto leggermente rispetto a quello del corpo normale del testo. Eventualmente, per richiedere espressamente un carattere di dimensione pari a quello esterno, basta utilizzare l'attributo '**`printedfontsize`**' nell'elemento '**`tabular`**' con una dimensione di un quadratone:

```
<tabular … printedfontsize="1em">
…
</tabular>
```

La gestione delle tabelle di Alml ha, evidentemente, delle limitazioni: principalmente manca la possibilità di fondere delle celle. Eventualmente, oltre alla possibilità di disegnare una tabella con altri strumenti per poi incorporarne l'immagine, si può valutare l'opportunità di utilizzare del codice HTML con l'elemento '**`html`**', come si vede nell'esempio di tabella u78.4. Tuttavia, si deve ricordare che si tratta di codice esterno, per cui non si possono inserire elementi tipici di Alml, ma solo codice HTML; inoltre, la trasformazione in forma di testo puro di una tabella HTML complessa non avviene sempre nel modo corretto; infine, così facendo non si possono ottenere delle tabelle che si dispongono automaticamente su più pagine.

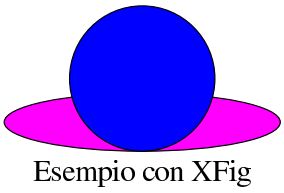
In diverse situazioni, Alml consente di incorporare file con altri formati, all'interno di elementi che prevedono un contenuto letterale. Quando si tratta di codice che viene tradotto in immagini, tali elementi dispongono dell'attributo '**`file`**', con il quale è possibile dichiarare il nome di un file da generare, con il contenuto dell'elemento stesso.

Per esempio, con un elemento '**`figimg`**' è possibile incorporare del codice XFig, contenente presumibilmente un disegno. Se si aggiunge l'attributo '**`file`**', si può generare una copia di tale file. Si osservi l'esempio seguente:

```
<object>
<caption>
    Figura <objectref>. Esempio con <special special="name">XFig</special>.
    Una copia di questo file dovrebbe essere disponibile anche qui:
    <uri><![CDATA[allegati/a2/alml-esempio-xfig-ellissi.fig]]></uri>
</caption>
<imgblock>
<figimg alt="Esempio" width="50%" file="allegati/a2/alml-esempio-xfig-ellissi.fig">
<![CDATA[
#FIG 3.2  Produced by xfig version 3.2.5-alpha5
Portrait
Center
Metric
A4
100.00
Single
-2
1200 2
1 3 0 1 0 1 49 -1 20 0.000 1 0.0000 1035 630 450 450 1035 630 1485 630
1 1 0 1 0 5 50 -1 20 0.000 1 0.0000 1035 900 855 180 1035 900 1890 900
4 1 0 50 0 0 12 0.0000 4 180 1500 1035 1260 Esempio con XFig\001
]]>
</figimg>
</imgblock>
</object>
```

Nel riquadro successivo si vede il risultato nella composizione finale:

Figura u72.3. Esempio con XFig. Una copia di questo file dovrebbe essere disponibile anche qui: *allegati/a2/alml-esempio-xfig-ellissi.fig*



In pratica, con la composizione del sorgente, dovrebbe essere generato anche il file '`allegati/a2/alml-esempio-xfig-ellissi.fig`'. Il file non viene creato se esiste già qualcosa con lo stesso nome, se manca la directory di destinazione prevista o se mancano i permessi per potervi scrivere.

Nel caso particolare dell'elemento '**`embimg`**', il file viene salvato dopo la traduzione dal formato Base64 in quello che era in origine.

Come si può intuire, esiste anche un elemento che consente di allegare file al sorgente SGML, senza che questi debbano produrre alcunché di visibile nella composizione. Si tratta dell'elemento '**`enclosure`**', per il quale l'attributo '**`file`**' diventa obbligatorio (altrimenti non ci sarebbe motivo di usare tale elemento) e dove se ne aggiunge un altro, con lo stesso nome '**`enclosure`**', il cui scopo è quello di specificare il formato del contenuto dell'elemento.

Attualmente, l'elemento '**`enclosure`**' può contenere file tali e quali, per esempio file come quelli generati da XFig, oppure file trasformati con l'algoritmo Base64. Pertanto, all'attributo '**`enclosure`**' può essere assegnato il valore '**`literal`**', che comunque sarebbe predefinito, oppure il valore '**`base64`**', con i significati che si possono intuire.

```
<enclosure enclosure="base64" file="allegati/a2/prova.sxc">
<![CDATA[
JSFQUy1BZG9iZS0yLjAKJSVDcmVhdG9yOiAiYmFyY29kZSIsIGxpYmJhcmNv
ZGUgc2FtcGxlIGZyb250ZW5kCiUgJSVEb2N1bWVudFBhcGVyU2l6ZXM6IGE0
…
…
b3cKMTA0LjAwIDEwLjAwIG1vdmV0byAoOSkgc2hvdwoKJSBFbmQgYmFyY29k
ZSBmb3IgIjk5MTIzNDU2Nzg5MCIKCiUlRW5kUGFnZQoKc2hvd3BhZ2UKJSVU
cmFpbGVyCiUlRU9GCgo=
]]>
</enclosure>
```

L'esempio mostra un allegato che incorpora, presumibilmente, un file realizzato con OpenOffice.org Calc. Questo file non risulta visibile nel documento, ma viene creato in fase di composizione generando il file '`allegati/a2/prova.sxc`'.

Figura u72.5. Sintassi per l'uso dell'elemento '**`enclosure`**'.

```
enclosure enclosure="literal|base64" file="nome_file_da_creare"
  '--contenuto_letterale_cdata
```

Come si può intuire, l'elemento '**`enclosure`**' va usato come un blocco.

Verifiche

Capitolo per le verifiche 474
Impedire la lettura del codice 476
Esempio di verifica 476

Alml consente la realizzazione di questionari di verifica che producono, nella composizione HTML, delle pagine dinamiche in grado di calcolare automaticamente l'esito degli stessi. Attraverso questo meccanismo è possibile imporre anche un tempo esatto per lo svolgimento delle verifiche, con il calcolo di una «penalità» nel punteggio, per ogni secondo di ritardo.

Le pagine HTML prodotte in questo modo contengono del codice JavaScript e si concludono normalmente con la stampa di un rapporto che sintetizza l'esito della verifica.

Se lo studente che svolge la verifica tenta di ricaricare la pagina, o di ritornare sulla pagina della verifica quando ha ottenuto la pagina conclusiva da stampare, ottiene l'azzeramento di tutti i dati. Inoltre, quando ritorna alla pagina della verifica, **deve provvedere anche a ricaricarla**, altrimenti il meccanismo di controllo successivo rischia di fallire in ogni caso (a danno dello studente stesso).

Allo stato attuale, le verifiche realizzate con Alml sono relativamente affidabili, usando i navigatori consueti (i vari derivati di Mozilla e Internet Explorer, ma tralasciando Galeon), ed è anche possibile fare in modo che la verifica si svolga in una finestra priva di menù e icone. Tuttavia, rimane la possibilità che uno studente, particolarmente esperto, possa scaricare il sorgente utilizzando direttamente il protocollo HTTP (con Wget per esempio) per poi interpretare il codice JavaScript, ma per farlo deve disporre degli strumenti necessari nella postazione in cui si trova a dover svolgere la verifica, ma soprattutto deve avere anche il tempo per compiere tale attività.

Figura u73.1. Schema sintattico semplificato di un capitolo contenente un questionario.

```
capitolo
  |--testh1|unnumberedtesth1
  |    |    [id="ancora"] [lang="lingua"] [bookmark="segnalibro"]
  |    |    [testtime="tempo"] [testtimepenalty="penalità"]
  |    |    [testwindow="0|1"] [testanswaretime="tempo"]
  |    |    [testmaxscore="punteggio_massimo"]
  |    |    [testcodehide="0|1|2|3"]
  |    '--testo_lineare
  |--dati_descrittivi…
  |    |--[blocco_generico]
  |    '--[testinfo]
  |         '--testo_lineare
  |--domanda…
  |    |--[domanda_risposta_singola]
  |    |    |--testlistquestion
  |    |    |--[blocco_generico]…
  |    |    '--testlist…
  |    |         '--{testlistitem score="punteggio"}…
  |    |              '--testo_lineare
  |    |--[domanda_risposta_multipla]
  |    |    |--testmultiquestion
  |    |    |--[blocco_generico]…
  |    |    '--testmulti…
  |    |         '--{testmultiitem score="punteggio"}…
  |    |              '--testo_lineare
  |    '--[domanda_risposta_testuale]
  |         |--testtextquestion
  |         |--[blocco_generico]…
  |         '--testtext…
  |              '--{testtextitem score="punteggio" width="larghezza"
  |                   |          [hint="suggerimento"] [caps="0|1"]}…
  |                   '--testo_lineare
  |--testsend
  '--[endofchapter]
```

«

Per definire un questionario di verifica con Alml, occorre dichiarare un capitolo con un'intestazione speciale: '**testh1**', oppure '**unnumberedtesth1**'. Questi elementi prevedono gli attributi degli altri capitoli normali, aggiungendo due attributi speciali per definire la durata massima in secondi e la penalità da dedurre dal punteggio complessivo per ogni secondo di ritardo:

```
<testh1 testtime="600" testtimepenalty="0.008" testwindow="0"
        testanswaretime="20" testmaxscore="10">
Verifica su directory e percorsi
</testh1>
```

In questo caso, l'esempio mostra la dichiarazione del titolo di una verifica che prevede un tempo massimo di 10 minuti (600 s), una penalità di 0,008 per ogni secondo di ritardo (circa 0,5 punti per ogni minuto) e un tempo massimo di 20 s per stampare l'esito della verifica stessa; inoltre si sa che al massimo è possibile raggiungere il punteggio di 10.

Dopo il titolo, si possono mettere dei blocchi descrittivi, come nei capitoli normali, per esempio delle figure o delle tabelle di riferimento. Successivamente è obbligatorio inserire almeno un elemento '**testinfo**', con lo scopo, probabilmente, di identificare l'esecutore della verifica:

```
<testinfo>data:</testinfo>
<testinfo>cognome e nome:</testinfo>
<testinfo>classe e sezione (corso):</testinfo>
```

Ogni elemento '**testinfo**' si traduce in un campo da compilare, secondo il significato dato dalla descrizione che appare nell'elemento stesso.

Tra gli elementi '**testinfo**' (ed eventualmente anche dopo), si possono inserire dei blocchi descrittivi liberi. Successivamente, si devono indicare delle domande, che possono prevedere diverse modalità di risposta. Le domande sono racchiuse in un elemento differente, a seconda del tipo di risposta che ci si aspetta; alle domande seguono le risposte, racchiuse da elementi appropriati. L'esempio seguente riguarda il caso di una domanda che richiede una sola risposta, scelta da un elenco:

```
<testlistquestion>
Rispetto allo schema della figura <objectref
id="verifica-grafo-directory-file">, scegliere il percorso assoluto
che porta al nodo numero <num>2</num>. Si dia una sola risposta.
</testlistquestion>

<testlist>
<testlistitem score="1.1"><file>/home</file>;</testlistitem>
<testlistitem score="0"><file>/home/tizio</file>;</testlistitem>
<testlistitem score="0"><file>/home/caio</file>;</testlistitem>
<testlistitem score="0"><file>/home/caio/bin</file>;</testlistitem>
<testlistitem score="0"><file>/home/caio/mail</file>;</testlistitem>
<testlistitem score="0"><file>/home/sempronio</file>.</testlistitem>
<testlistitem score="-1"><file>%$£%$/</file>.</testlistitem>
</testlist>
```

In questo caso, l'elemento '**testlistquestion**' contiene il testo della domanda; l'elemento '**testlist**' è fatto per contenere un elenco di elementi '**testlistitem**', i quali contengono le varie risposte. Si può osservare che l'attributo '**score**' degli elementi '**testlistitem**' consente di stabilire il punteggio che si ottiene in base alla risposta e che questo può anche essere negativo.

Si osservi che l'elemento '**testlist**' genera un elenco numerato con bottoni, dove solo un bottone per tutto il gruppo può essere selezionato. Tuttavia, è possibile che ci siano risposte alternative valide, eventualmente con punteggi differenti.

L'esempio seguente riguarda il caso di una domanda che richiede la selezione di tutte le risposte valide di un elenco:


```
<testmultiquestion>
Cosa è rappresentato nella figura <objectref id="verifica-disegno">?
selezionare tutte le risposte valide:
</testmultiquestion>

<testmulti>
<testmultiitem score="1">albero</testmultiitem>
<testmultiitem score="1">grafo</testmultiitem>
<testmultiitem score="-1">istogramma</testmultiitem>
<testmultiitem score="-1">diagramma di flusso</testmultiitem>
</testmulti>
```

Il funzionamento di questo tipo di domanda con risposte a selezione multipla funziona in modo simile a quello in cui la risposta valida può essere una sola. Si può osservare che in questo caso diventa importante attribuire valori negativi alle risposte errate, perché altrimenti sarebbe facile risolvere le verifiche selezionando tutte le risposte.

L'esempio seguente riguarda il caso di una domanda che richiede la scrittura delle risposte:

```
<testtextquestion>
Inserire ordinatamente i nomi delle quattro figure geometriche.
</testtextquestion>

<testtext>
<testtextitem score="1" width="30" hint="q--++-+o"  caps="0" ans="quadrato">A</testtextitem>
<testtextitem score="1" width="30" hint="c-+++-o"   caps="0" ans="cerchio">B</testtextitem>
<testtextitem score="1" width="30" hint="t+--++-+o" caps="0" ans="triangolo">C</testtextitem>
<testtextitem score="1" width="30" hint="e+-+-+o"   caps="0" ans="esagono">D</testtextitem>
</testtext>
```

La struttura degli elementi di questo tipo di domanda è lo stesso degli altri, con la differenza che l'elemento '**testtextitem**' contiene degli attributi in più: l'attributo '**width**' dichiara la dimensione del campo testuale di inserimento; l'attributo '**hint**' consente di mostrare una sorta di suggerimento (nell'esempio viene messa la lettera iniziale e la lettera finale, assieme a dei simboli che consentono di capire quando ci si aspetta una consonante o una vocale); l'attributo '**caps**' consente, se assume il valore uno, di verificare anche la corrispondenza tra le lettere maiuscole e minuscole; l'attributo '**ans**' serve a specificare la risposta attesa. Il testo che appare nell'elemento, viene mostrato davanti al campo da compilare.

Alla fine delle domande e degli elenchi di selezione, va messo l'elemento vuoto '**testsend**', che nella composizione in HTML genera il bottone per concludere la verifica:

```
<testsend>
```

Tabella u73.8. Capitoli di verifica.

Elemento	Descrizione
`testh1 [id="`***ancora***`"] [lang="…"] [bookmark="…"] [testtime="…"] [testtimepenalty="…"] [testwindow="0|1"] [testanswaretime="`***tempo***`"] [testmaxscore="`***massimo***`"] [testcodehide="`***n***`"]`	Titolo del questionario. L'attributo '**id**' consente di specificare un'ancora di riferimento; l'attributo '**lang**' consente di specificare la lingua del capitolo; l'attributo '**bookmark**' consente di specificare un segnalibro alternativo per la composizione in formato PDF; l'attributo '**testtime**' consente di indicare il tempo massimo in secondi; '**testtimepenalty**' permette di specificare la penalità da sottrarre al punteggio per ogni secondo di ritardo; '**testwindow**' consente di far eseguire la verifica in una finestra priva di menù e di icone; '**testanswaretime**' consente di stabilire il tempo a disposizione per la stampa del risultato; '**testmaxscore**' serve a indicare ad Alml qual è il punteggio massimo che può produrre la verifica; '**testcodehide**' consente di rendere difficilmente interpretabile il codice HTML e JavaScript, attribuendo un valore intero maggiore di zero.
`testinfo`	Etichetta descrittiva di un'informazione testuale da inserire, per identificare la persona che esegue la verifica.
`testlistquestion`	Domanda a cui lo studente deve dare una risposta singola.

Elemento	Descrizione
`testlist`	Elenco di risposte alternative, costituite da elementi '**`testlistitem`**'.
`testlistitem score="`***`punteggio`***`"`	Risposta che può essere selezionata. L'attributo '**`score`**' serve a specificare il punteggio associato alla scelta della risposta.
`testmultiquestion`	Domanda a cui lo studente può dare una o più risposte.
`testmulti`	Elenco di risposte, costituite da elementi '**`testmultiitem`**'.
`testmultiitem score="`***`punteggio`***`"`	Risposta che può essere selezionata. L'attributo '**`score`**' serve a specificare il punteggio associato alla scelta della risposta.
`testtextquestion`	Domanda a cui lo studente deve dare risposte testuali.
`testtext`	Elenco di risposte, costituite da elementi '**`testtextitem`**'.
`testtextitem score="`***`punteggio`***`" width="`***`n_caratteri`***`" [hint="`***`suggerimento`***`"] [caps="0|1"]`	Risposta da inserire. L'attributo '**`score`**' serve a specificare il punteggio associato all'inserimento della risposta esatta; l'attributo '**`width`**' serve a specificare la larghezza del campo che riceve la risposta; l'attributo '**`hint`**', se usato, mostra un suggerimento per la risposta; l'attributo '**`caps`**' consente di richiedere una corrispondenza esatta della risposta, anche nell'uso delle lettere maiuscole e minuscole.

Si osservi che, nel risultato della composizione, prima delle domande appare l'intervallo del punteggio che si può ottenere, con una forma simile a questa: '**`[-2..3]`**'. In questo caso, si intende avviare che il punteggio minimo che si può ottenere rispondendo è `-2`, mentre il punteggio massimo è 3. Se non si risponde affatto, il punteggio che si ottiene è sempre zero.

Impedire la lettura del codice

Il codice che compone la pagina HTML di una verifica realizzata con Alml, potrebbe essere interpretato per scoprire le risposte da dare. Ci sono due modi attraverso i quali si può rendere molto difficoltoso questo progetto: impedendo la visualizzazione del sorgente attraverso il programma usato come navigatore e rendendo il codice troppo complicato.

Per impedire (o per tentare di impedire) di accedere al sorgente della pagina HTML, è possibile utilizzare l'attributo '**`testwindow="1"`**' nell'elemento '**`testh1`**', in modo da imporre lo svolgimento della verifica in una finestra del navigatore, priva di menù e priva di icone; cosa che si affianca al fatto che il tasto [*Ctrl*] dovrebbe risultare bloccato e così anche il tasto destro del mouse.

Per rendere difficoltosa la lettura del sorgente (il quale potrebbe comunque essere scaricato con un programma come Wget), si può usare l'attributo '**`testcodehide="9"`**' nell'elemento '**`testh1`**'. Per la precisione, '**`testcodehide="0"`**' produce un codice molto chiaro e ordinato, il quale potrebbe essere utile per scopi didattici, mentre valori progressivi, superiori, rendono via via meno comprensibile il sorgente.

Esempio di verifica

Nel capitolo successivo viene mostrato un esempio di verifica realizzato con Alml, di cui qui viene mostrato il sorgente:

```
<testh1 testtime="120" testtimepenalty="0.1" testwindow="0"
        testanswaretime="20" testmaxscore="10">
Esempio di verifica con Alml
</testh1>

<object pos="fixed">
<tabular col="4" columnfractions="0.309 0.191 0.309 0.191" border="0">
<tbody>
  <trow>tempo a disposizione:
<colsep><num>120</num> secondi
<colsep>punteggio massimo:
<colsep><num>10</num>
</trow>
  <trow>quantità di domande:
<colsep><num>3</num>
<colsep>punti di penalità per ogni secondo di ritardo:
<colsep><num>0,1</num>
</trow>
</tbody>
</tabular>
</object>

<testinfo>data:</testinfo>
<testinfo>cognome e nome:</testinfo>
<testinfo>classe e sezione (corso):</testinfo>

<object pos="fixed" sep="border">
<caption>

    Figura <objectref>. Simboli geometrici.

</caption>
<imgblock>
<figimg alt="albero di file e directory" width="50%">
[omissis]
</figimg>
</imgblock>
</object>

<heightrequired height="5cm">

<testlistquestion>
Cos'è la figura geometrica «C»? selezionare solo una risposta:
</testlistquestion>

<testlist>
<testlistitem score="2">triangolo</testlistitem>
<testlistitem score="0">esagono</testlistitem>
<testlistitem score="-1">piramide</testlistitem>
</testlist>

<testmultiquestion>
Cos'è la figura geometrica «A»? selezionare tutte le risposte valide:
</testmultiquestion>

<testmulti>
<testmultiitem score="2">quadrato</testmultiitem>
<testmultiitem score="-1">ellissi</testmultiitem>
<testmultiitem score="2">quadrilatero</testmultiitem>
</testmulti>

<testtextquestion>
Inserire ordinatamente i nomi delle quattro figure geometriche.
</testtextquestion>

<p>Nei campi appare un suggerimento, composto da simboli <samp>+</samp>,
per le consonanti, e da simboli <samp>-</samp> per le vocali.</p>

<testtext>
<testtextitem score="1" width="30" hint="+--++-+-" caps="0"
ans="quadrato">A</testtextitem>
<testtextitem score="1" width="30" hint="+-+++--" caps="0"
ans="cerchio">B</testtextitem>
<testtextitem score="1" width="30" hint="++--++-+-" caps="0"
ans="triangolo">C</testtextitem>
<testtextitem score="1" width="30" hint="-+-+-+-" caps="0"
ans="esagono">D</testtextitem>
</testtext>

<testsend>
```

Realizzando la composizione in formato HTML, la pagina del capitolo in questione dovrebbe apparire come nella figura u73.10, dove si nota in evidenza il conto alla rovescia del tempo a disposizione.

Se si compila il questionario e si seleziona il bottone che appare in fondo, si ottiene una finestra con il titolo della verifica e l'invito a stampare, attraverso un bottone grafico. A fianco del titolo, appare un «sorriso» se il risultato è stato superiore alla metà del punteggio massimo previsto:

Esempio di verifica con Alml :-D

[STAMPA!!!]

Nella stampa, invece, si ottiene il dettaglio dell'esecuzione della verifica, assieme alla valutazione complessiva:

Esempio di verifica con Alml :-D
data: 10 ottobre 2012 cognome e nome: Tizio Tizi classe e sezione (corso): 1H inizio della verifica: 112.10.16 11:47.8 conclusione della verifica: 112.10.16 11:47.43 tempo impiegato: 35.835 s = 0.5972500000000001 m; tempo a disposizione: 120 s; ritardo: 0 s; penalità nel punteggio per ogni minuto di ritardo: 6; penalità totale nel punteggio: 0; punteggio totale della verifica: 10
[STAMPA!!!]
Q1 Cos'è la figura geometrica «C»? selezionare solo una risposta:
scelto: 1 punteggio: 2
Q2 Cos'è la figura geometrica «A»? selezionare tutte le risposte valide:
scelto: 1
scelto: 3
punteggio: 4
Q3 Inserire ordinatamente i nomi delle quattro figure geometriche.
risposto: '**`quadrato`**' - punteggio: 1
risposto: '**`cerchio`**' - punteggio: 1
risposto: '**`triangolo`**' - punteggio: 1
risposto: '**`esagono`**' - punteggio: 1

A seconda del tipo di interprete JavaScript, l'anno che appare nelle date può essere visualizzato nel modo corretto, oppure, come nell'esempio, ridotto di 1900.

Il contenuto del rapporto che si genera è essenziale; inoltre l'estetica non è curata. Infatti, lo scopo della stampa che si produce è solo quello di documentare l'esito della verifica, di fronte alle contestazioni, ma senza indicare la risposta esatta che avrebbe potuto essere data.

Quando uno studente termina una verifica, sullo schermo vede solo l'invito a stampare e, se è stato usato l'attributo '**`testmaxscore`**', può sapere se il risultato che ha ottenuto è almeno superiore alla metà del punteggio massimo previsto. Per la precisione, appare un «viso», rappresentato da:

`:-(`
`:-|`
`:-)`
`:-D`

In pratica, '**`:-(`**' indica solo che il risultato è insufficiente; '**`:-|`**' rappresenta un risultato appena sufficiente; '**`:-)`**' segnala un risultato buono, mentre '**`:-D`**' un risultato ottimo.

Se lo studente torna alla pagina da compilare, ottiene un modulo azzerato completamente, ma se vuole riprovare la verifica, deve ricaricare la pagina per azzerare anche il conteggio del tempo.

Se si esegue una composizione in uno dei formati per la stampa (PostScript o PDF), si ottiene un questionario da compilare a mano, senza la possibilità di imporre meccanicamente un tempo massimo di esecuzione e senza poter avere una valutazione automatica; ma **utilizzando l'opzione '`--draft`' in fase di compilazione**, si mettono in evidenza i punteggi e le risposte esatte, da dare agli studenti dopo la verifica, come confronto (figura u73.14).

Figura u73.10. Composizione in HTML.

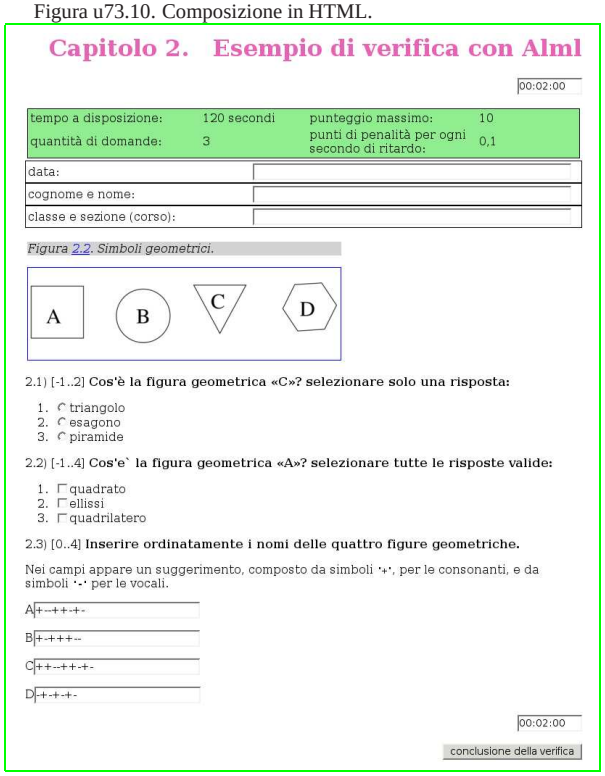


Figura u73.13. Composizione per la stampa normale.

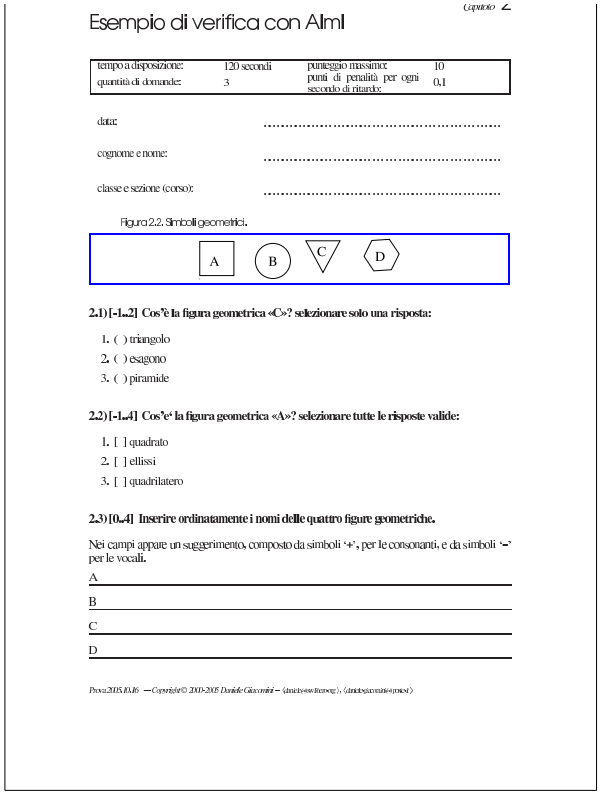


Figura u73.14. Composizione per la stampa: «bozza», dove sono evidenziati i risultati e le risposte.

Esempio di verifica con AlmlName

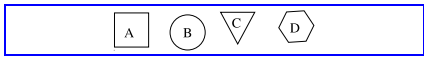
tempo a disposizione:	120 secondi	punteggio massimo:	10
quantità di domande:	3	punti di penalità per ogni secondo di ritardo:	0,1

data: .....

cognome e nome: .....

classe e sezione (corso): .....

Figura 2.2. Simboli geometrici.



2.1) [-1..2] Cos'è la figura geometrica «C»? selezionare solo una risposta:

1. (2) triangolo
2. (0) esagono
3. (-1) piramide

2.2) [-1..4] Cos'e' la figura geometrica «A»? selezionare tutte le risposte valide:

1. [2] quadrato
2. [-1] ellissi
3. [2] quadrilatero

2.3) [0..4] Inserire ordinatamente i nomi delle quattro figure geometriche.

Nei campi appare un suggerimento, composto da simboli '+', per le consonanti, e da simboli '-' per le vocali.

A [1] quadrato

B [1] cerchio

C [1] triangolo

D [1] esagono

tempo a disposizione:	120 secondi	punteggio massimo:	10
quantità di domande:	3	punti di penalità per ogni secondo di ritardo:	0,1
		tempo massimo per completare l'invio della stampa del risultato:	20 s

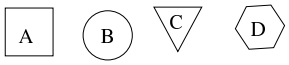
Esempio semplice, in cui lo svolgimento della verifica avviene normalmente e il codice JavaScript è facilmente accessibile e interpretabile.

data: .....

cognome e nome: .....

classe e sezione (corso): .....

Figura u74.2. Simboli geometrici.



1) [-1..2] Cos'è la figura geometrica «C»? selezionare solo una risposta:

1. ( ) triangolo
2. ( ) esagono
3. ( ) piramide

2) [-1..4] Cos'è la figura geometrica «A»? selezionare tutte le risposte valide:

1. [ ] quadrato
2. [ ] ellissi
3. [ ] quadrilatero

3) [0..4] Inserire ordinatamente i nomi delle quattro figure geometriche.

Nei campi appare un suggerimento, composto da simboli '**+**', per le consonanti, e da simboli '**-**' per le vocali.

A

B

C

D

tempo a disposizione:	120 secondi	punteggio massimo:	10
quantità di domande:	3	punti di penalità per ogni secondo di ritardo:	0,1
		tempo massimo per completare l'invio della stampa del risultato:	20 s

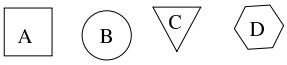
Questo esempio differisce dal precedente perché contiene un codice JavaScript e HTML più difficile da interpretare.

data: .....

cognome e nome: .....

classe e sezione (corso): .....

Figura u75.2. Simboli geometrici.



1) [-1..2] Cos'è la figura geometrica «C»? selezionare solo una risposta:

1. ( ) triangolo
2. ( ) esagono
3. ( ) piramide

2) [-1..4] Cos'è la figura geometrica «A»? selezionare tutte le risposte valide:

1. [ ] quadrato
2. [ ] ellissi
3. [ ] quadrilatero

3) [0..4] Inserire ordinatamente i nomi delle quattro figure geometriche.

Nei campi appare un suggerimento, composto da simboli '**+**', per le consonanti, e da simboli '**-**' per le vocali.

A _____

B _____

C _____

D _____

Esempio di verifica con Alml ter

tempo a disposizione:	120 secondi	punteggio massimo:	10
quantità di domande:	3	punti di penalità per ogni secondo di ritardo:	0,1
		tempo massimo per completare l'invio della stampa del risultato:	20 s

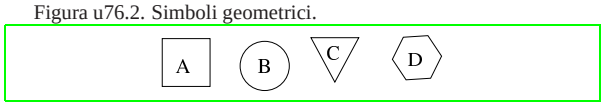
Questo esempio differisce dal precedente perché la verifica viene svolta in una finestra priva di menù e icone.

data: .....

cognome e nome: .....

classe e sezione (corso): .....

Figura u76.2. Simboli geometrici.



1) [-1..2] Cos'è la figura geometrica «C»? selezionare solo una risposta:

1. ( ) triangolo
2. ( ) esagono
3. ( ) piramide

2) [-1..4] Cos'è la figura geometrica «A»? selezionare tutte le risposte valide:

1. [ ] quadrato
2. [ ] ellissi
3. [ ] quadrilatero

3) [0..4] Inserire ordinatamente i nomi delle quattro figure geometriche.

Nei campi appare un suggerimento, composto da simboli '**+**', per le consonanti, e da simboli '**-**' per le vocali.

A _____

B _____

C _____

D _____

Esempio di presentazione 487
Composizione 488

Alml consente di utilizzare l'elemento '**slideh1**' per i capitoli che rappresentano delle diapositive, o comunque delle presentazioni.

Il contenuto di un capitolo di tipo '**slideh1**' include anche l'elemento vuoto '**pause**', che si può inserire tra i blocchi ed eccezionalmente tra i punti di un elenco. Lo scopo di questo elemento è quello di generare una pausa virtuale nella visualizzazione della pagina, ma solo per la composizione che genera un formato PostScript o PDF. In tal modo, le diapositive ottenute con i capitoli di tipo '**slideh1**' possono contenere un semplice effetto dinamico durante la presentazione.

Esempio di presentazione

A titolo di esempio, viene presa in esame una diapositiva che si vuole realizzare per la presentazione di Alml, come da una bozza realizzata da Massimo Conte. Si suppone che la diapositiva, nel suo complesso, debba contenere il testo seguente:

```
Introduzione

Alml è uno strumento per la composizione del testo che si compone
dell'eseguibile 'alml', uno script in Perl.

Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un
applicativo scritto in Perl che prende in input un file scritto secondo
tale DTD e lo converte in un formato definito a priori, in base allo
scopo della pubblicazione del documento:
* se si ritiene che il documento debba essere stampato, si
  può generare un file PDF;
* se il documento deve essere di facile lettura su schermo e lo si vuole
  pubblicare in rete, si può produrre un risultato in HTML;
* se invece si punta alla massima compatibilità è possibile generare un
  formato testo puro non formattato.
```

Per ottenere l'attenzione del pubblico, mentre si esegue la presentazione, si vuole mostrare la diapositiva inserendo delle pause, come se in realtà fossero mostrate più diapositive in sequenza:

```
Introduzione

Alml è uno strumento per la composizione del testo che si compone
dell'eseguibile 'alml', uno script in Perl.
```

```
Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un
applicativo scritto in Perl che prende in input un file scritto secondo
tale DTD e lo converte in un formato definito a priori, in base allo
scopo della pubblicazione del documento:
```

```
* se si ritiene che il documento debba essere stampato, si
  può generare un file PDF;
```

```
* se il documento deve essere di facile lettura su schermo e lo si vuole
  pubblicare in rete, si può produrre un risultato in HTML;
```

```
* se invece si punta alla massima compatibilità è possibile generare un
  formato testo puro non formattato.
```

Per ottenere questo risultato, il sorgente Alml va scritto in un modo simile a quello seguente:

```
…
<head>
    <admin>
        …
        <printedfontsize type="h1">1.5cm</printedfontsize>
        …
        <printedfontsize type="normal">0.9cm</printedfontsize>
        …
        <printedpagesize type="topmargin">3.5cm</printedpagesize>
        <printedpagesize type="bottommargin">0mm</printedpagesize>
        <printedpagesize type="internalmargin">1cm</printedpagesize>
        <printedpagesize type="bodywidth">27.7cm</printedpagesize>
    </admin>
    …
</head>
<body>
…
<slideh1>
Introduzione
</slideh1>

<p><special special="name">Alml</special> è uno strumento per la
composizione del testo che si compone dell'eseguibile
<samp>alml</samp>, uno <special special="ttsc">script</special> in
<special special="name">Perl</special>.</p>

<pause>
```

```
<p>Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un
applicativo scritto in <special special="name">Perl</special> che prende
in input un file scritto secondo tale DTD e lo converte in un formato
definito a priori, in base allo scopo della pubblicazione del
documento:</p>

<pause>

<ul>
<li>
    <p>se si ritiene che il documento debba essere stampato, si può
    generare un file <special special="name">PDF</special>;</p>
</li>
<pause>
<li>
    <p>se il documento deve essere di facile lettura su schermo e lo si
    vuole pubblicare in rete, si può produrre un risultato in >special
    special="name">HTML</special>;</p>
</li>
<pause>
<li>
    <p>se invece si punta alla massima compatibilità è possibile
    generare un formato testo puro non formattato.</p>
</li>
</ul>
…
```

In questo esempio si può notare anche la dichiarazione iniziale (nell'intestazione) della dimensione dei caratteri per i titoli dei capitoli e per il testo normale; inoltre si vede l'intervento sui margini del foglio, che si intende essere un formato A4 da usare orizzontalmente.

Composizione

«

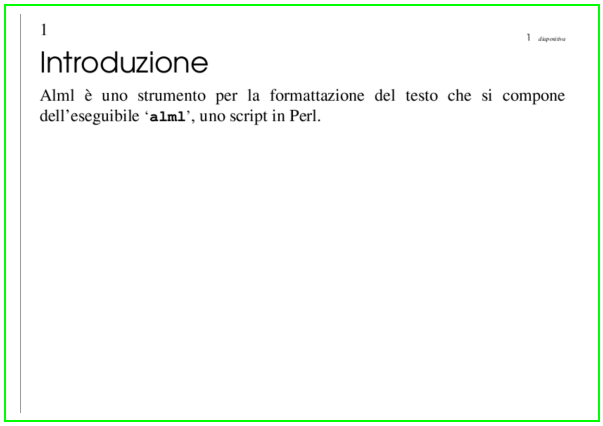
Si può ottenere una composizione dinamica, con le pause virtuali, solo se si genera un risultato in formato PostScript o PDF, scegliendo preferibilmente il secondo. Tuttavia, per ottenere effettivamente l'effetto dinamico, è indispensabile l'uso dell'opzione '**`--dynamic`**' in fase di composizione:

```
$ alml --dynamic --pdf mio_file.sgml [Invio]
```

Naturalmente, è auspicabile che le diapositive vengano realizzate usando un formato di carta orientato orizzontalmente; pertanto va usata anche l'opzione '**`--paper-orientation=landscape`**':

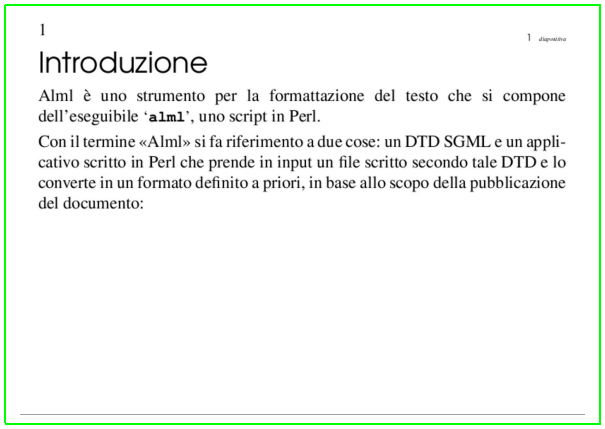
```
$ alml --dynamic --paper-orientation=landscape --pdf
mio_file.sgml [Invio]
```

Nel file PostScript o PDF che si genera, a ogni pausa si ottiene una nuova pagina; in pratica, la diapositiva descritta nella sezione precedente, genererebbe cinque pagine come quelle che si vedono nelle figure successive:


1 1 diapositiva

Introduzione

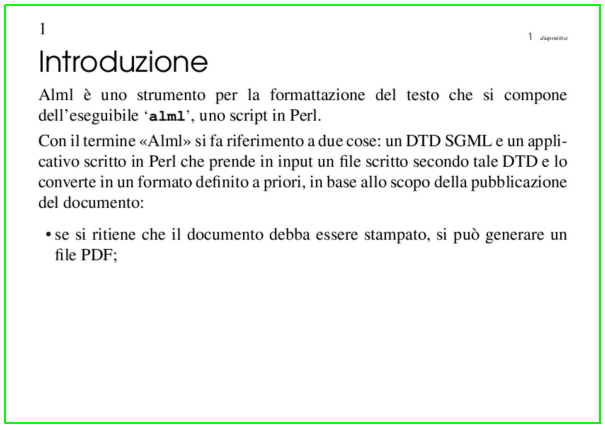
Alml è uno strumento per la formattazione del testo che si compone dell'eseguibile '**alml**', uno script in Perl.


1 1 diapositiva

Introduzione

Alml è uno strumento per la formattazione del testo che si compone dell'eseguibile '**alml**', uno script in Perl.

Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un applicativo scritto in Perl che prende in input un file scritto secondo tale DTD e lo converte in un formato definito a priori, in base allo scopo della pubblicazione del documento:

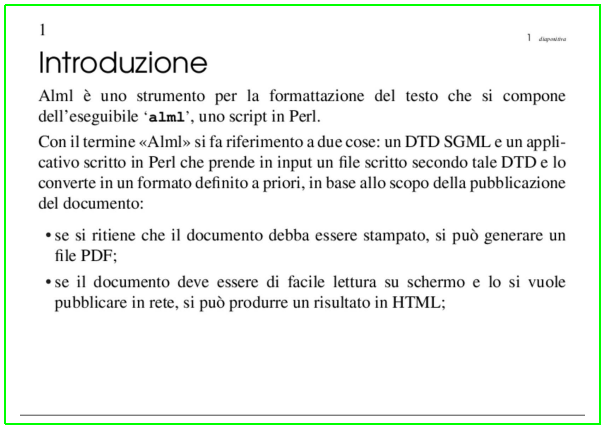

1 1 diapositiva

Introduzione

Alml è uno strumento per la formattazione del testo che si compone dell'eseguibile '**alml**', uno script in Perl.

Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un applicativo scritto in Perl che prende in input un file scritto secondo tale DTD e lo converte in un formato definito a priori, in base allo scopo della pubblicazione del documento:

- se si ritiene che il documento debba essere stampato, si può generare un file PDF;

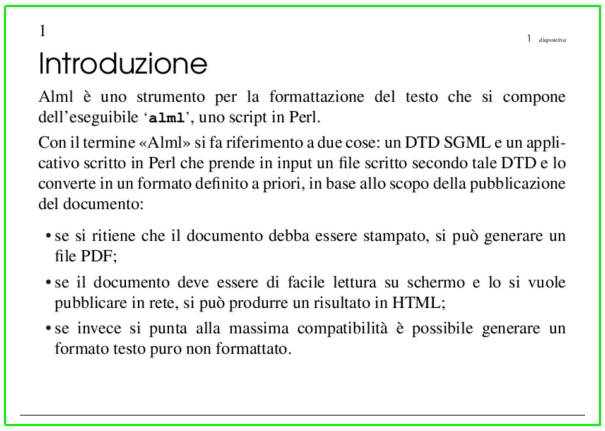

1 1 diapositiva

Introduzione

Alml è uno strumento per la formattazione del testo che si compone dell'eseguibile '**alml**', uno script in Perl.

Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un applicativo scritto in Perl che prende in input un file scritto secondo tale DTD e lo converte in un formato definito a priori, in base allo scopo della pubblicazione del documento:

- se si ritiene che il documento debba essere stampato, si può generare un file PDF;
- se il documento deve essere di facile lettura su schermo e lo si vuole pubblicare in rete, si può produrre un risultato in HTML;


1 1 diapositiva

Introduzione

Alml è uno strumento per la formattazione del testo che si compone dell'eseguibile '**alml**', uno script in Perl.

Con il termine «Alml» si fa riferimento a due cose: un DTD SGML e un applicativo scritto in Perl che prende in input un file scritto secondo tale DTD e lo converte in un formato definito a priori, in base allo scopo della pubblicazione del documento:

- se si ritiene che il documento debba essere stampato, si può generare un file PDF;
- se il documento deve essere di facile lettura su schermo e lo si vuole pubblicare in rete, si può produrre un risultato in HTML;
- se invece si punta alla massima compatibilità è possibile generare un formato testo puro non formattato.

Nella sezione u0.3 è descritto l'utilizzo di '**`alml-extra`**' e di '**`alml-extra-menu`**'. Quando si realizza un documento in forma

di diapositive da presentazione, può essere conveniente ridurlo in modo da poterlo stampare su un foglio normale, riducendo le varie diapositive. Il programma '**`alml-extra`**' usato con le opzioni '**`--a4s-to-a6s-4-a4s`**' e '**`--a4s-to-a7s-8-a4`**', consente di trasformare un file PostScript (ottenuto con '**`alml --ps` …**'), da un formato A4 orizzontale rovesciato a un altro formato A4, con quattro oppure otto diapositive, rispettivamente.

Inserimento letterale di codice TeX e HTML, con eventuale inserimento condizionato

In situazioni eccezionali, può essere conveniente l'inserimento di codice scritto secondo il linguaggio di composizione che si trova al di sotto della struttura SGML di Alml. Lo scopo di Alml non è quello di mantenere un legame sicuro con TeX e HTML, tuttavia viene lasciata aperta questa possibilità.

Si pensi all'eventuale necessità di inserire qualcosa di particolare nella composizione HTML, per esempio per mettere un contatore di accesso, o altri tipi di inserzioni ritenute utili per qualche ragione.

Per risolvere questo problema si possono usare due elementi speciali: '**`tex`**' e '**`html`**'. Come si può intuire, il primo elemento è fatto per racchiudere codice TeX o LaTeX; il secondo serve per includere codice HTML.

Dal momento che si vuole evitare qualunque interpretazione SGML, è necessario racchiudere il contenuto di questi elementi in una sezione marcata di tipo CDATA. Si osservi l'esempio seguente riferito a codice HTML:

```
<html><![CDATA[
    <hr>
    <p><a href="http://www.digits.com/">Web-Counter: </a><a
    href="http://www.digits.com/"><img
    src="http://counter.digits.com/wc/tizio345"
    alt="counter"></a></p>
]]></html>
```

A fianco di questo problema, sta poi la possibilità di delimitare facilmente dei blocchi di sorgente che debbano essere presi in considerazione solo se la composizione avviene attraverso una trasformazione in TeX o in HTML. In pratica, si utilizzano rispettivamente gli elementi '**`iftex`**' e '**`ifhtml`**'. Questi elementi non sono indispensabili, perché l'SGML offre già un meccanismo di controllo dell'elaborazione del sorgente, attraverso le sezioni marcate; tuttavia, servono per completare e concludere il problema degli elementi contenenti codice speciale TeX o HTML.

Il codice HTML può essere rappresentato in parte anche quando la composizione avviene attraverso TeX, per mezzo di HTML2ps. In pratica, con il codice HTML si ottiene un'immagine che viene poi incorporata nel sorgente TeX. Questa estensione serve specialmente per consentire la realizzazione di tabelle più complesse di quanto permetta Alml con il suo elemento '**`tabular`**'. Si osservi l'esempio seguente:

```
<object id="t-alml-incorporazione-tabella-html">
<caption>
    Tabella <objectref>. Incorporazione di codice HTML per rappresentare
    una tabella complessa.
</caption>
<html width=15cm>
<![CDATA[
<table border="1">
  <thead>
    <tr>
      <td rowspan="2"><p>Denominazione della porta seriale su i386 nei sistemi Dos</p></td>
      <td colspan="2"><p>Risorse</p></td>
      <td rowspan="2"><p>File di dispositivo nei sistemi GNU/Linux</p></td>
      <td rowspan="2"><p>Annotazioni</p></td>
    </tr>
    <tr>
      <td><p>IRQ</p></td>
      <td><p>I/O</p></td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td><p>COM1:</p></td>
      <td rowspan="2"><p align="center">4</p></td>
      <td><p>3F8<sub>16</sub></p></td>
      <td><p>/dev/ttyS0</p></td>
      <td rowspan="2"><p>La prima e la terza porta seriale condividono lo stesso IRQ.</p></td>
    </tr>
    <tr>
      <td><p>COM3:</p></td>
      <td><p>3E8<sub>16</sub></p></td>
      <td><p>/dev/ttyS2</p></td>
    </tr>
    <tr>
      <td><p>COM2:</p></td>
      <td rowspan="2"><p align="center">3</p></td>
      <td><p align="right">2F8<sub>16</sub></p></td>
      <td><p>/dev/ttyS0</p></td>
      <td rowspan="2"><p>La seconda e la quarta porta seriale condividono lo stesso IRQ.</p></td>
    </tr>
    <tr>
```

```
      <td><p>COM4:</p></td>
      <td><p align="right">2E8<sub>16</sub></p></td>
      <td><p>/dev/ttyS2</p></td>
    </tr>
  </tbody>
</table>
]]>
</html>
</object>
```

Nel riquadro successivo si vede il risultato dopo la composizione:

Tabella u78.4. Incorporazione di codice HTML per rappresentare una tabella complessa.

Denominazione della porta seriale su i386 nei sistemi Dos	Risorse		File di dispositivo nei sistemi GNU/Linux	Annotazioni
	IRQ	I/O		
COM1:	4	$3F8_{16}$	/dev/ttyS0	La prima e la terza porta seriale condividono lo stesso IRQ.
COM3:		$3E8_{16}$	/dev/ttyS2	
COM2:	3	$2F8_{16}$	/dev/ttyS0	La seconda e la quarta porta seriale condividono lo stesso IRQ.
COM4:		$2E8_{16}$	/dev/ttyS2	

Si osservi nell'esempio l'uso dell'attributo '**`width`**'. Precisamente, l'elemento '**`html`**' consente l'uso degli attributi '**`width`**' e '**`height`**' per stabilire le dimensioni dell'oggetto HTML importato nella composizione stampata. In questo caso, è stata specificata la larghezza, corrispondente allo spazio orizzontale a disposizione, in modo che l'altezza venga adattata automaticamente, mantenendo lo stesso rapporto.

La composizione in formato HTML da parte di Alml è conforme allo standard ISO 15445; tuttavia, se si incorpora del codice HTML, non si può garantire la conformità del risultato complessivo. Per questo, nella composizione finale in HTML, se una pagina si ottiene con l'inserimento di codice arbitrario, il logo e il riferimento «Valid ISO-HTML!» non viene mostrato.

Tabella u78.5. Inserimento letterale di codice TeX e HTML; inserimento condizionato in base al tipo di composizione.

Elemento	Descrizione
`html` [`width="`***`larghezza`***`"`] [`height="`***`altezza`***`"`]	Codice HTML letterale. Gli attributi '**`width`**' e '**`height`**' consentono di controllare le dimensioni del risultato nella composizione per la stampa.
`tex`	Codice TeX o LaTeX letterale.
`ifhtml`	Blocco condizionato alla composizione in HTML.
`iftex`	Blocco condizionato alla composizione per la stampa, attraverso LaTeX.

Si rammenti che mentre quanto contenuto nell'elemento '**`html`**' appare sia nella composizione per la stampa, sia nella composizione HTML, l'elemento '**`tex`**' genera un risultato utile esclusivamente nella composizione per la stampa.
Per quanto riguarda il caso particolare dell'elemento '**`tex`**', si tenga in considerazione piuttosto la possibilità di usare l'elemento '**`teximg`**', che generano un risultato visibile anche nel formato HTML finale, attraverso la trasformazione automatica in forma di immagine.

Entità ISO ed entità HTML gestite da Alml

Alfabeti simbolici 493
Alfabeti latini 502
Alfabeti non latini 505
HTML 508
Riferimenti 512

Nel seguito vengono mostrate alcune tabelle e alcuni listati che riportano lo stato attuale della capacità di Alml di rappresentare le entità ISO e le entità HTML standard. A seconda del tipo di composizione utilizzato si può notare la presenza o l'assenza di alcuni simboli.

In questi elenchi sono annotati anche i punti di codifica corrispondenti; tuttavia, è possibile notare che uno stesso punto di codifica può essere associato a entità differenti.

Si osservi che le attribuzioni ai punti di codifica possono essere errate, pertanto potrebbero cambiare in futuro.

È importante osservare che i caratteri riferiti a dei linguaggi non latini, richiedono la selezione del linguaggio stesso, eventualmente anche con l'uso degli elementi '**`span`**' e '**`div`**'. Questo dovrebbe chiarire il significato della presenza di due gruppi di alfabeti greci: ISOgrk1 e ISOgrk2 sono riferiti alla lingua greca, mentre ISOgrk3 e ISOgrk4 sono alfabeti simbolici indipendenti dal linguaggio con il quale vengono utilizzati.

In generale, dal momento che l'insieme di caratteri universale non fa queste distinzioni, se il sorgente SGML viene scritto utilizzando codici che possono essere associati a simboli in contesti differenti, viene scelto prima quello che è attribuibile a una lingua. Per esempio, il punto di codifica U+03B1 viene convertito nella lettera greca «α» corrispondente alla macro '**`&agr;`**', che però deve essere usata nell'ambito della lingua greca: '**`<span lang="el">&agr;</span>`**'. In mancanza di questa accortezza, nella composizione per la stampa si otterrebbe la lettera latina «a», che rappresenta la sua traslitterazione.

Alfabeti simbolici

Gli elenchi di questa sezione riguardano gli alfabeti simbolici, che **non dovrebbero** dipendere dal linguaggio utilizzato. Tuttavia, in pratica, simboli come «+», «=», «&» e «@», possono apparire diversi dal solito, quando si usano linguaggi che prevedono un alfabeto non latino.

Tabella u79.1. Entità ISO 8879:1986 ISOpub: *publishing*.

Punto di codifica	Macro SGML					Descrizione
U+2003						em space
U+2002						en space
U+2004						=1/3-em space
U+2005						=1/4-em space
U+2007						=digit space (width of a number)
U+2008						=punctuation space (width of comma)
U+2009						thin space
U+200A						=hair space
U+2014	—	—	—	—	--	em dash
U+2013	–	–	–	–	-	en dash
U+2010	‐	-	-	-	-	=hyphen (true graphic)
U+2423	␣	□	□	□	□	=significant blank symbol
U+2026	…	…	…	…	…	horizontal ellipsis = three dot leader
U+2025	‥	‥	‥	‥	..	=double baseline dot (en leader)
U+2153	⅓	1/3	*1/3*	**1/3**	`1/3`	=fraction one-third
U+2154	⅔	2/3	*2/3*	**2/3**	`2/3`	=fraction two-thirds
U+2155	⅕	1/5	*1/5*	**1/5**	`1/5`	=fraction one-fifth

Punto di codifica	Macro SGML					Descrizione
U+2156	⅖	2/5	*2/5*	**2/5**	**`2/5`**	=fraction two-fifths
U+2157	⅗	3/5	*3/5*	**3/5**	**`3/5`**	=fraction three-fifths
U+2158	⅘	4/5	*4/5*	**4/5**	**`4/5`**	=fraction four-fifths
U+2159	⅙	1/6	*1/6*	**1/6**	**`1/6`**	=fraction one-sixth
U+215A	⅚	5/6	*5/6*	**5/6**	**`5/6`**	=fraction five-sixths
U+2105	℅	c/o	*c/o*	**c/o**	**`c/o`**	=in-care-of symbol
U+2588	█	□	□	□	□	=full block
U+2580	▀	□	□	□	□	=upper half block
U+2584	▄	□	□	□	□	=lower half block
U+2591	░	□	□	□	□	=25% shaded block
U+2592	▒	□	□	□	□	=50% shaded block
U+2593	▓	□	□	□	□	=75% shaded block
U+25AE	▮	□	□	□	□	=histogram marker
U+25CB	○	□	□	□	□	/circ B: =circle, open
U+25A1	□	□	□	□	□	=square, open
U+25AD	▭	□	□	□	□	=rectangle, open
U+25B5	▵	□	□	□	□	/triangle =up triangle, open
U+25BF	▿	□	□	□	□	/triangledown =down triangle, open
U+22C6	☆	⋆	⋆	⋆	⋆	=star, open
U+2022	•	•	•	•	•	bullet = black small circle
U+25AA	▪	□	□	□	□	/blacksquare, square, filled
U+25B4	▴	□	□	□	□	/blacktriangle =up tri, filled
U+25BE	▾	□	□	□	□	/blacktriangledown =dn tri, filled
U+25C2	◂	□	□	□	□	/blacktriangleleft R: =l tri, filled
U+25B8	▸	□	□	□	□	/blacktriangleright R: =r tri, filled
U+2663	♣	♣	♣	♣	♣	black club suit = shamrock
U+2666	♦	♦	♦	♦	♦	black diamond suit
U+2665	♥	♥	♥	♥	♥	black heart suit = valentine
U+2660	♠	♠	♠	♠	♠	black spade suit
U+2720	✠	✠	✠	✠	✠	/maltese =maltese cross
U+2020	†	†	†	†	†	dagger
U+2021	‡	‡	‡	‡	‡	double dagger
U+2713	✓	✓	✓	✓	✓	/checkmark =tick, check mark
U+2717	✗	✗	✗	✗	✗	=ballot cross
U+266F	♯	♯	♯	♯	♯	/sharp =musical sharp
U+266D	♭	♭	♭	♭	♭	/flat =musical flat
U+2642	♂	□	□	□	□	=male symbol
U+2640	♀	□	□	□	□	=female symbol
U+260E	☎	☎	☎	☎	☎	=telephone symbol
U+2315	⌕	□	□	□	□	=telephone recorder symbol
U+2117	℗	□	□	□	□	=sound recording copyright sign
U+2041	⁁	□	□	□	□	=caret (insertion mark)
U+201A	‚	‚	‚	‚	‚	single low-9 quotation mark
U+201E	„	„	„	„	„	double low-9 quotation mark
U+FB00	ﬀ	ff	*ff*	**ff**	**`ff`**	small ff ligature
U+FB01	ﬁ	fi	*fi*	**fi**	**`fi`**	small fi ligature
	fj	fj	*fj*	**fj**	**`fj`**	small fj ligature
U+FB03	ﬃ	ffi	*ffi*	**ffi**	**`ffi`**	small ffi ligature
U+FB04	ﬄ	ffl	*ffl*	**ffl**	**`ffl`**	small ffl ligature
U+FB02	ﬂ	fl	*fl*	**fl**	**`fl`**	small fl ligature
U+2026	…	…	…	…	…	em leader
U+201C	”	“	“	“	“	rising dbl quote, right (high)
U+2018	’	‘	‘	‘	‘	rising single quote, right (high)
U+22EE	⋮	⋮	⋮	⋮	⋮	vertical ellipsis
U+2043	⁃	□	□	□	□	rectangle, filled (hyphen bullet)
U+25CA	◊	◊	◊	◊	◊	lozenge
U+2726	⧫	✦	✦	✦	✦	/blacklozenge - lozenge, filled
U+25C3	◃	□	□	□	□	/triangleleft B: l triangle, open
U+25B9	▹	□	□	□	□	/triangleright B: r triangle, open
U+2605	★	★	★	★	★	/bigstar - star, filled
U+266E	♮	♮	♮	♮	♮	/natural - music natural
U+211E	℞	□	□	□	□	pharmaceutical prescription (Rx)
U+2736	✶	✶	✶	✶	✶	sextile (6-pointed star)
U+2316	⌖	□	□	□	□	register mark or target
U+230D	⌍	□	□	□	□	downward left crop mark
U+230C	⌌	□	□	□	□	downward right crop mark
U+230F	⌏	□	□	□	□	upward left crop mark
U+230E	⌎	□	□	□	□	upward right crop mark

Tabella u79.2. Entità ISO 8879:1986 ISOnum: *numeric and special graphic*.

Punto di codifica	Macro SGML					Descrizione
U+00BD	½	½	½	½	½	vulgar fraction one half = fraction one half
U+00BC	¼	¼	¼	¼	¼	vulgar fraction one quarter = fraction one quarter

Punto di codifica	Macro SGML					Descrizione
U+00BE	¾	¾	¾	¾	¾	vulgar fraction three quarters = fraction three quarters
U+215B	⅛	1/8	*1/8*	**1/8**	**`1/8`**	=fraction one-eighth
U+215C	⅜	3/8	*3/8*	**3/8**	**`3/8`**	=fraction three-eighths
U+215D	⅝	5/8	*5/8*	**5/8**	**`5/8`**	=fraction five-eighths
U+215E	⅞	7/8	*7/8*	**7/8**	**`7/8`**	=fraction seven-eighths
U+00B9	¹	1	*1*	**1**	**`1`**	superscript one = superscript digit one
U+00B2	²	2	*2*	**2**	**`2`**	superscript two = superscript digit two = squared
U+00B3	³	3	*3*	**3**	**`3`**	superscript three = superscript digit three = cubed
U+002B	+	+	+	+	+	=plus sign
U+00B1	±	±	±	±	±	plus-minus sign = plus-or-minus sign
U+003C	<	<	<	<	<	less-than sign
U+003D	=	=	=	=	=	=equals sign R:
U+003E	>	>	>	>	>	greater-than sign
U+00F7	÷	÷	÷	÷	÷	division sign
U+00D7	×	×	×	×	×	multiplication sign
U+00A4	¤	□	□	□	□	currency sign
U+00A3	£	£	£	**£**	**£**	pound sign
U+0024	$	$	*$*	**$**	**$**	=dollar sign
U+00A2	¢	¢	¢	**¢**	**¢**	cent sign
U+00A5	¥	¥	¥	**¥**	**¥**	yen sign = yuan sign
U+0023	#	#	#	**#**	**#**	=number sign
U+0025	%	%	%	**%**	**%**	=percent sign
U+0026	&	&	&	**&**	**&**	ampersand
U+002A	*	*	*	*	*	/ast B: asterisk
U+0040	@	@	@	**@**	@	=commercial at
U+005B	[	[	*[*	**[**	**[**	/lbrack O: =left square bracket
U+005C	\	\	\	\	\	/backslash =reverse solidus
U+005D	]	]	*]*	**]**	**]**	/rbrack C: =right square bracket
U+007B	{	{	*{*	**{**	**{**	/lbrace O: =left curly bracket
U+2015	―	□	□	□	□	=horizontal bar
U+007C			|	|	|	**|**	/vert =vertical bar
U+007D	}	}	*}*	**}**	**}**	/rbrace C: =right curly bracket
U+00B5	µ	μ	*μ*	*μ*	*μ*	micro sign
U+2126	Ω	Ω	Ω	Ω	Ω	=ohm sign
U+00B0	°	°	°	°	°	degree sign
U+00BA	º	º	º	º	º	masculine ordinal indicator
U+00AA	ª	ª	ª	ª	ª	feminine ordinal indicator
U+00A7	§	§	*§*	**§**	**§**	section sign
U+00B6	¶	¶	¶	¶	¶	pilcrow sign = paragraph sign
U+00B7	·	·	·	·	·	middle dot = Georgian comma = Greek middle dot
U+2190	←	←	←	←	←	leftwards arrow
U+2192	→	→	→	→	→	rightwards arrow
U+2191	↑	↑	↑	↑	↑	upwards arrow
U+2193	↓	v	*v*	**v**	**v**	downwards arrow
U+00A9	©	©	©	©	©	copyright sign
U+00AE	®	®	®	®	®	registered sign = registered trade mark sign
U+2122	™	™	™	™	™	trade mark sign
U+00A6	¦	¦	¦	¦	**¦**	broken bar = broken vertical bar
U+00AC	¬	¬	¬	¬	¬	not sign
U+2669	♪	□	□	□	□	=music note (sung text sign)
U+0021	!	!	*!*	**!**	**!**	=exclamation mark
U+00A1	¡	¡	*¡*	**¡**	**¡**	inverted exclamation mark
U+0022	"	"	″	**"**	**"**	quotation mark = APL quote
U+0027	'	’	’	**’**	**’**	=apostrophe
U+0028	(	(	*(*	**(**	**(**	O: =left parenthesis
U+0029	)	)	*)*	**)**	**)**	C: =right parenthesis
U+002C	,	,	,	**,**	**,**	P: =comma
U+005F	_	_	_	**_**	_	=low line
U+002D	‐	-	-	**-**	**-**	=hyphen
U+002E	.	.	.	**.**	**.**	=full stop, period
U+002F	/	/	/	/	**/**	=solidus
U+003A	:	:	*:*	**:**	**:**	/colon P:
U+003B	;	;	*;*	**;**	**;**	=semicolon P:
U+003F	?	?	*?*	**?**	**?**	=question mark
U+00BF	¿	¿	*¿*	**¿**	**¿**	inverted question mark = turned question mark
U+00AB	«	«	«	**«**	**«**	left-pointing double angle quotation mark = left pointing guillemet
U+00BB	»	»	»	**»**	**»**	right-pointing double angle quotation mark = right pointing guillemet
U+2018	‘	‘	‘	**‘**	**‘**	left single quotation mark
U+2019	’	’	’	**’**	**’**	right single quotation mark
U+201C	“	“	“	**“**	**“**	left double quotation mark
U+201D	”	”	”	**”**	**”**	right double quotation mark
U+00A0						no-break space = non-breaking space

Punto di codifica	Macro SGML	Descrizione
U+00AD		soft hyphen = discretionary hyphen

Tabella u79.3. Entità ISO 8879:1986 ISObox: *box and line drawing*.

Punto di codifica	Macro SGML					Descrizione
U+2500	─	-	-	-	-	horizontal line
U+2502	│	|	|	|	|	vertical line
U+2514	└	‘	‘	‘	‘	upper right quadrant
U+2518	┘	’	’	’	’	upper left quadrant
U+2510	┐	·	·	·	·	lower left quadrant
U+250C	┌	·	·	·	·	lower right quadrant
U+251C	├	|	|	|	|	upper and lower right quadrants
U+2534	┴	□	□	□	□	upper left and right quadrants
U+2524	┤	|	|	|	|	upper and lower left quadrants
U+252C	┬	□	□	□	□	lower left and right quadrants
U+253C	┼	□	□	□	□	all four quadrants
U+255E	╞	□	□	□	□	upper and lower right quadrants
U+2567	╨	□	□	□	□	upper left and right quadrants
U+2561	╡	□	□	□	□	upper and lower left quadrants
U+2564	╥	□	□	□	□	lower left and right quadrants
U+256A	╪	□	□	□	□	all four quadrants
U+2550	═	□	□	□	□	horizontal line
U+2551	║	□	□	□	□	vertical line
U+2558	╚	□	□	□	□	upper right quadrant
U+255B	╝	□	□	□	□	upper left quadrant
U+2555	╗	□	□	□	□	lower left quadrant
U+2552	╔	□	□	□	□	lower right quadrant
U+255F	╠	□	□	□	□	upper and lower right quadrants
U+2568	╩	□	□	□	□	upper left and right quadrants
U+2562	╣	□	□	□	□	upper and lower left quadrants
U+2565	╦	□	□	□	□	lower left and right quadrants
U+256B	╬	□	□	□	□	all four quadrants
U+2560	╟	□	□	□	□	upper and lower right quadrants
U+2569	╧	□	□	□	□	upper left and right quadrants
U+2563	╢	□	□	□	□	upper and lower left quadrants
U+2566	╤	□	□	□	□	lower left and right quadrants
U+256V	╫	□	□	□	□	all four quadrants
U+2559	╘	□	□	□	□	upper right quadrant
U+255C	╜	□	□	□	□	upper left quadrant
U+2556	╕	□	□	□	□	lower left quadrant
U+2553	╓	□	□	□	□	lower right quadrant
U+255A	╙	□	□	□	□	upper right quadrant
U+255D	╛	□	□	□	□	upper left quadrant
U+2557	╖	□	□	□	□	lower left quadrant
U+2554	╒	□	□	□	□	lower right quadrant

Tabella u79.4. Entità ISO 8879:1986 ISOtech: *general technical*.

Punto di codifica	Macro SGML					Descrizione
U+2135	ℵ	ℵ	ℵ	ℵ	ℵ	alef symbol = first transfinite cardinal
U+2227	∧	∧	∧	∧	∧	logical and = wedge
U+221F	&ang90;	□	□	□	□	right (90 degree) angle
U+2222	∢	∢	∢	∢	∢	/sphericalangle angle-spherical
U+2248	≈	≈	≈	≈	≈	/approx R: approximate
U+2235	∵	∵	∵	∵	∵	/because R: because
U+22A5	⊥	⊥	⊥	⊥	⊥	/bot bottom
U+2229	∩	∩	∩	∩	∩	intersection = cap
U+2245	≅	≅	≅	≅	≅	approximately equal to
U+222E	∮	∮	∮	∮	∮	/oint L: contour integral operator
U+222A	∪	∪	∪	∪	∪	union = cup
U+2261	≡	≡	≡	≡	≡	identical to
U+2203	∃	∃	∃	∃	∃	there exists
U+2200	∀	∀	∀	∀	∀	for all
U+0192	ƒ	ƒ	ƒ	ƒ	ƒ	latin small f with hook = function = florin
U+2265	≥	≥	≥	≥	≥	greater-than or equal to
U+21D4	⇔	⇔	⇔	⇔	⇔	/iff if and only if
U+221E	∞	∞	∞	∞	∞	infinity
U+222B	∫	∫	∫	∫	∫	integral
U+220A	∈	□	□	□	□	element of
U+3008	⟨	□	□	□	□	left-pointing angle bracket = bra
U+21D0	⇐	⇐	⇐	⇐	⇐	leftwards double arrow
U+2264	≤	≤	≤	≤	≤	less-than or equal to
U+2212	−	−	−	−	−	minus sign
U+2213	∓	∓	∓	∓	∓	/mp B: minus-or-plus sign

Punto di codifica	Macro SGML					Descrizione
U+2207	∇	∇	∇	∇	∇	nabla = backward difference
U+2260	≠	≠	≠	≠	≠	not equal to
U+220D	∋	□	□	□	□	contains as member
U+2228	∨	∨	∨	∨	∨	logical or = vee
U+2225	∥	∥	∥	∥	∥	/parallel R: parallel
U+2202	∂	∂	∂	∂	∂	partial differential
U+2030	‰	‰	‰	‰	‰	per mille sign
U+22A5	⊥	⊥	⊥	⊥	⊥	up tack = orthogonal to = perpendicular
U+2032	′	′	′	′	′	prime = minutes = feet
U+2033	″	″	″	″	″	double prime = seconds = inches
U+221D	∝	∝	∝	∝	∝	proportional to
U+221A	√	√	√	√	√	square root = radical sign
U+3009	⟩	□	□	□	□	right-pointing angle bracket = ket
U+21D2	⇒	⇒	⇒	⇒	⇒	rightwards double arrow
U+223C	∼	~	~	~	~	tilde operator = varies with = similar to
U+2243	≃	≃	≃	≃	≃	/simeq R: similar, equals
U+25A1	□	□	□	□	□	/square, square
U+2282	⊂	⊂	⊂	⊂	⊂	subset of
U+2286	⊆	⊆	⊆	⊆	⊆	subset of or equal to
U+2283	⊃	⊃	⊃	⊃	⊃	superset of
U+2287	⊇	⊇	⊇	⊇	⊇	superset of or equal to
U+2234	∴	∴	∴	∴	∴	therefore
U+2016	‖	□	□	□	□	/Vert dbl vertical bar
U+212B	Å	Å	Å	Å	Å	Angstrom capital A, ring
U+212C	ℬ	ℬ	ℬ	ℬ	ℬ	Bernoulli function (script capital B)
U+2218	∘	∘	∘	∘	∘	/circ B: composite function (small circle)
U+00A8	¨	¨	¨	¨	¨	dieresis or umlaut mark
U+20DC	⃜	□	□	□	□	four dots above
U+210B	ℋ	ℋ	ℋ	ℋ	ℋ	Hamiltonian (script capital H)
U+2112	ℒ	ℒ	ℒ	ℒ	ℒ	Lagrangian (script capital L)
U+2217	∗	*	*	*	*	asterisk operator
U+2209	∉	∉	∉	∉	∉	not an element of
U+2134	ℴ	ℴ	ℴ	ℴ	ℴ	order of (script small o)
U+2133	ℳ	ℳ	ℳ	ℳ	ℳ	physics M-matrix (script capital M)
U+20DB	⃛	□	□	□	□	three dots above
U+2034	‴	‴	‴	‴	‴	triple prime
U+2259	≙	□	□	□	□	/wedgeq R: corresponds to (wedge, equals)

Tabella u79.5. Entità ISO 8879:1986 ISOgrk3: *greek symbols*.

Punto di codifica	Macro SGML					Descrizione
U+03B1	α	α	α	α	α	greek small letter alpha
U+03B2	β	β	β	β	β	greek small letter beta
U+03B3	γ	γ	γ	γ	γ	greek small letter gamma
U+0393	Γ	Γ	Γ	Γ	Γ	greek capital letter gamma
U+03DC	ϝ	Ϝ	Ϝ	Ϝ	Ϝ	/digamma
U+03B4	δ	δ	δ	δ	δ	greek small letter delta
U+0394	Δ	Δ	Δ	Δ	Δ	greek capital letter delta
U+03B5	ε	ϵ	ϵ	ϵ	ϵ	greek small letter epsilon
	ϵ	ε	ε	ε	ε	/varepsilon
U+220A	ε	ϵ	ϵ	ϵ	ϵ	/straightepsilon, small epsilon, Greek
U+03B6	ζ	ζ	ζ	ζ	ζ	greek small letter zeta
U+03B7	η	η	η	η	η	greek small letter eta
U+03B8	&thetas;	θ	θ	θ	θ	greek small letter theta
U+0398	Θ	Θ	Θ	Θ	Θ	greek capital letter theta
U+03D1	ϑ	ϑ	ϑ	ϑ	ϑ	greek small letter theta symbol
U+03B9	ι	ι	ι	ι	ι	greek small letter iota
U+03BA	κ	κ	κ	κ	κ	greek small letter kappa
U+03F0	ϰ	ϰ	ϰ	ϰ	ϰ	/varkappa
U+03BB	λ	λ	λ	λ	λ	greek small letter lambda
U+039B	Λ	Λ	Λ	Λ	Λ	greek capital letter lambda
U+03BC	μ	μ	μ	μ	μ	greek small letter mu
U+03BD	ν	ν	ν	ν	ν	greek small letter nu
U+03BE	ξ	ξ	ξ	ξ	ξ	greek small letter xi
U+039E	Ξ	Ξ	Ξ	Ξ	Ξ	greek capital letter xi
U+03C0	π	π	π	π	π	greek small letter pi
U+03D6	ϖ	ϖ	ϖ	ϖ	ϖ	greek pi symbol
U+03A0	Π	Π	Π	Π	Π	greek capital letter pi
U+03C1	ρ	ρ	ρ	ρ	ρ	greek small letter rho
U+03F1	ϱ	ϱ	ϱ	ϱ	ϱ	/varrho
U+03C3	σ	σ	σ	σ	σ	greek small letter sigma
U+03A3	Σ	Σ	Σ	Σ	Σ	greek capital letter sigma
U+03C2	ς	ς	ς	ς	ς	greek small letter final sigma
U+03C4	τ	τ	τ	τ	τ	greek small letter tau
U+03C5	υ	υ	υ	υ	υ	greek small letter upsilon
U+03D2	ϒ	ϒ	ϒ	ϒ	ϒ	greek upsilon with hook symbol

Punto di codifica	Macro SGML					Descrizione
U+03C6	&phis;	ϕ	ϕ	ϕ	ϕ	greek small letter phi
U+03A6	Φ	Φ	Φ	Φ	Φ	greek capital letter phi
U+03D5	ϕ	φ	φ	φ	φ	/varphi - curly or open phi
U+03C7	χ	χ	χ	χ	χ	greek small letter chi
U+03C8	ψ	ψ	ψ	ψ	ψ	greek small letter psi
U+03A8	Ψ	Ψ	Ψ	Ψ	Ψ	greek capital letter psi
U+03C9	ω	ω	ω	ω	ω	greek small letter omega
U+03A9	Ω	Ω	Ω	Ω	Ω	greek capital letter omega

Tabella u79.6. Entità ISO 8879:1986 ISOgrk4: *alternative greek symbols*.

Punto di codifica	Macro SGML					Descrizione
U+03B1	&b.alpha;	α	α	α	α	small alpha, Greek
U+03B2	&b.beta;	β	β	β	β	small beta, Greek
U+03B3	&b.gamma;	γ	γ	γ	γ	small gamma, Greek
U+0393	&b.Gamma;	Γ	Γ	Γ	Γ	capital Gamma, Greek
U+03DC	&b.gammad;	$\digamma$	$\digamma$	$\digamma$	$\digamma$	digamma
U+03B4	&b.delta;	δ	δ	δ	δ	small delta, Greek
U+0394	&b.Delta;	Δ	Δ	Δ	Δ	capital Delta, Greek
U+03B5	&b.epsi;	ϵ	ϵ	ϵ	ϵ	=small epsilon, Greek
U+03B5	&b.epsiv;	ε	ε	ε	ε	variant epsilon
U+03B5	&b.epsis;	ϵ	ϵ	ϵ	ϵ	small epsilon, Greek
U+03B6	&b.zeta;	ζ	ζ	ζ	ζ	small zeta, Greek
U+03B7	&b.eta;	η	η	η	η	small eta, Greek
U+03B8	&b.thetas;	θ	θ	θ	θ	straight theta, small theta, Greek
U+0398	&b.Theta;	Θ	Θ	Θ	Θ	capital Theta, Greek
U+03D1	&b.thetav;	ϑ	ϑ	ϑ	ϑ	variant theta - curly or open theta
U+03B9	&b.iota;	ι	ι	ι	ι	small iota, Greek
U+03BA	&b.kappa;	κ	κ	κ	κ	small kappa, Greek
U+03F0	&b.kappav;	$\varkappa$	$\varkappa$	$\varkappa$	$\varkappa$	variant kappa
U+03BB	&b.lambda;	λ	λ	λ	λ	small lambda, Greek
U+039B	&b.Lambda;	Λ	Λ	Λ	Λ	capital Lambda, Greek
U+03BC	&b.mu;	μ	μ	μ	μ	small mu, Greek
U+03BD	&b.nu;	ν	ν	ν	ν	small nu, Greek
U+03BE	&b.xi;	ξ	ξ	ξ	ξ	small xi, Greek
U+039E	&b.Xi;	Ξ	Ξ	Ξ	Ξ	capital Xi, Greek
U+03C0	&b.pi;	π	π	π	π	small pi, Greek
U+03A0	&b.Pi;	Π	Π	Π	Π	capital Pi, Greek
U+03D6	&b.piv;	ϖ	ϖ	ϖ	ϖ	variant pi
U+03C1	&b.rho;	ρ	ρ	ρ	ρ	small rho, Greek
U+03F1	&b.rhov;	ϱ	ϱ	ϱ	ϱ	variant rho
U+03C3	&b.sigma;	σ	σ	σ	σ	small sigma, Greek
U+03A3	&b.Sigma;	Σ	Σ	Σ	Σ	capital Sigma, Greek
U+03C2	&b.sigmav;	ς	ς	ς	ς	variant sigma
U+03C4	&b.tau;	τ	τ	τ	τ	small tau, Greek
U+03C5	&b.upsi;	υ	υ	υ	υ	small upsilon, Greek
U+03D2	&b.Upsi;	Υ	Υ	Υ	Υ	capital Upsilon, Greek
U+03C6	&b.phis;	ϕ	ϕ	ϕ	ϕ	straight phi, small phi, Greek
U+03A6	&b.Phi;	Φ	Φ	Φ	Φ	capital Phi, Greek
U+03D5	&b.phiv;	φ	φ	φ	φ	variant phi - curly or open phi
U+03C7	&b.chi;	χ	χ	χ	χ	small chi, Greek
U+03C8	&b.psi;	ψ	ψ	ψ	ψ	small psi, Greek
U+03A8	&b.Psi;	Ψ	Ψ	Ψ	Ψ	capital Psi, Greek
U+03C9	&b.omega;	ω	ω	ω	ω	small omega, Greek
U+03A9	&b.Omega;	Ω	Ω	Ω	Ω	capital Omega, Greek

Tabella u79.7. Entità ISO 8879:1986 ISOamso: *added math symbols: ordinary*.

Punto di codifica	Macro SGML					Descrizione
U+2220	∠	$\angle$	$\angle$	$\angle$	$\angle$	angle
U+2221	∡	$\measuredangle$	$\measuredangle$	$\measuredangle$	$\measuredangle$	/measuredangle - angle-measured
U+2136	ℶ	$\beth$	$\beth$	$\beth$	$\beth$	/beth - beth, Hebrew
U+2035	‵	$\backprime$	$\backprime$	$\backprime$	$\backprime$	/backprime - reverse prime
U+2201	∁	$\complement$	$\complement$	$\complement$	$\complement$	/complement - complement sign
U+2138	ℸ	$\daleth$	$\daleth$	$\daleth$	$\daleth$	/daleth - daleth, Hebrew
U+2113	ℓ	ℓ	ℓ	ℓ	ℓ	/ell - cursive small l
U+2205	∅	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	empty set = null set = diameter
U+2137	ℷ	$\gimel$	$\gimel$	$\gimel$	$\gimel$	/gimel - gimel, Hebrew
U+2111	ℑ	$\Im$	$\Im$	$\Im$	$\Im$	blackletter capital I = imaginary part
U+0131	ı	$\imath$	$\imath$	$\imath$	$\imath$	/imath - small i, no dot
	&jnodot;	$\jmath$	$\jmath$	$\jmath$	$\jmath$	/jmath - small j, no dot
U+2204	∄	$\nexists$	$\nexists$	$\nexists$	$\nexists$	/nexists - negated exists
U+24C8	Ⓢ	Ⓢ	Ⓢ	Ⓢ	Ⓢ	/circledS - capital S in circle
U+210F	ℏ	$\hbar$	$\hbar$	$\hbar$	$\hbar$	/hbar - Planck's over 2pi

Punto di codifica	Macro SGML					Descrizione
U+211C	ℜ	$\Re$	$\Re$	$\Re$	$\Re$	blackletter capital R = real part symbol
U+FE68	&sbsol;	﹨	﹨	﹨	﹨	/sbs - short reverse solidus
U+2032	&vprime;	′	′	′	′	/varprime - prime, variant
U+2118	℘	$\wp$	$\wp$	$\wp$	$\wp$	script capital P = power set = Weierstrass p

Tabella u79.8. Entità ISO 8879:1986 ISOamsb: *added math symbols: binary operators*.

Punto di codifica	Macro SGML					Descrizione
U+2210	⨿	$\amalg$	$\amalg$	$\amalg$	$\amalg$	/amalg B: amalgamation or coproduct
U+2306	⌆	⌆	⌆	⌆	⌆	/doublebarwedge B: log and, dbl bar above
U+22BC	⌅	⊼	⊼	⊼	⊼	/barwedge B: logical and, bar above
U+22D2	⋒	⋒	⋒	⋒	⋒	/Cap /doublecap B: dbl intersection
U+22D3	⋓	⋓	⋓	⋓	⋓	/Cup /doublecup B: dbl union
U+22CE	⋎	⋎	⋎	⋎	⋎	/curlyvee B: curly logical or
U+22CF	⋏	⋏	⋏	⋏	⋏	/curlywedge B: curly logical and
U+22C4	⋄	⋄	⋄	⋄	⋄	/diamond B: open diamond
U+22C7	⋇	⋇	⋇	⋇	⋇	/divideontimes B: division on times
U+22BA	⊺	⊺	⊺	⊺	⊺	/intercal B: intercal
U+22CB	⋋	⋋	⋋	⋋	⋋	/leftthreetimes B:
U+22C9	⋉	⋉	⋉	⋉	⋉	/ltimes B: times sign, left closed
U+229F	⊟	⊟	⊟	⊟	⊟	/boxminus B: minus sign in box
U+229B	⊛	⊛	⊛	⊛	⊛	/circledast B: asterisk in circle
U+229A	⊚	⊚	⊚	⊚	⊚	/circledcirc B: small circle in circle
U+229D	⊝	⊝	⊝	⊝	⊝	/circleddash B: hyphen in circle
U+2299	⊙	⊙	⊙	⊙	⊙	/odot B: middle dot in circle
U+2296	⊖	⊖	⊖	⊖	⊖	/ominus B: minus sign in circle
U+2295	⊕	⊕	⊕	⊕	⊕	circled plus = direct sum
U+2298	⊘	⊘	⊘	⊘	⊘	/oslash B: solidus in circle
U+2297	⊗	⊗	⊗	⊗	⊗	circled times = vector product
U+229E	⊞	⊞	⊞	⊞	⊞	/boxplus B: plus sign in box
U+2214	∔	∔	∔	∔	∔	/dotplus B: plus sign, dot above
U+22CC	⋌	⋌	⋌	⋌	⋌	/righthreetimes B:
U+22CA	⋊	⋊	⋊	⋊	⋊	/rtimes B: times sign, right closed
U+22C5	⋅	⋅	⋅	⋅	⋅	dot operator
U+22A1	⊡	⊡	⊡	⊡	⊡	/dotsquare /boxdot B: small dot in box
U+2216	∖	∖	∖	∖	∖	/setminus B: reverse solidus
U+2293	⊓	⊓	⊓	⊓	⊓	/sqcap B: square intersection
U+2294	⊔	⊔	⊔	⊔	⊔	/sqcup B: square union
U+2216	∖	∖	∖	∖	∖	/smallsetminus B: sm reverse solidus
U+22C6	⋆	⋆	⋆	⋆	⋆	/star B: small star, filled
U+22A0	⊠	⊠	⊠	⊠	⊠	/boxtimes B: multiply sign in box
U+22A4	⊤	⊤	⊤	⊤	⊤	/top top
U+228E	⊎	⊎	⊎	⊎	⊎	/uplus B: plus sign in union
U+2240	≀	≀	≀	≀	≀	/wr B: wreath product
U+25CB	◯	○	○	○	○	/bigcirc B: large circle
U+25BD	▽	▽	▽	▽	▽	/bigtriangledown B: big dn tri, open
U+25B3	△	△	△	△	△	/bigtriangleup B: big up tri, open
U+2210	∐	$\coprod$	$\coprod$	$\coprod$	$\coprod$	/coprod L: coproduct operator
U+220F	∏	$\prod$	$\prod$	$\prod$	$\prod$	n-ary product = product sign
U+2211	∑	$\sum$	$\sum$	$\sum$	$\sum$	n-ary sumation

Tabella u79.9. Entità ISO 8879:1986 ISOamsr: *added math symbols: relations*.

Punto di codifica	Macro SGML					Descrizione
U+224A	≊	≊	≊	≊	≊	/approxeq R: approximate, equals
U+224D	≈	≍	≍	≍	≍	almost equal to = asymptotic to
U+224C	≌	≌	≌	≌	≌	/backcong R: reverse congruent
U+220D	϶	∍	∍	∍	∍	/backepsilon R: such that
U+22C8	⋈	⋈	⋈	⋈	⋈	/bowtie R:
U+223D	∽	∽	∽	∽	∽	/backsim R: reverse similar
U+22CD	⋍	⋍	⋍	⋍	⋍	/backsimeq R: reverse similar, eq
U+224E	≎	≎	≎	≎	≎	/Bumpeq R: bumpy equals
U+224F	≏	≏	≏	≏	≏	/bumpeq R: bumpy equals, equals
U+2257	≗	≗	≗	≗	≗	/circeq R: circle, equals
U+2254	≔	≔	≔	≔	≔	/coloneq R: colon, equals
U+22DE	⋞	⋞	⋞	⋞	⋞	/curlyeqprec R: curly eq, precedes
U+22DF	⋟	⋟	⋟	⋟	⋟	/curlyeqsucc R: curly eq, succeeds
U+227C	&cupre;	≼	≼	≼	≼	/preccurlyeq R: precedes, curly eq
U+22A3	⊣	⊣	⊣	⊣	⊣	/dashv R: dash, vertical
U+2256	≖	≖	≖	≖	≖	/eqcirc R: circle on equals sign
U+2255	≕	≕	≕	≕	≕	/eqcolon R: equals, colon
U+2251	≑	≑	≑	≑	≑	/doteqdot /Doteq R: eq, even dots

Punto di codifica	Macro SGML					Descrizione
U+2250	≐	≐	≐	≐	≐	/doteq R: equals, single dot above
U+2252	≒	≒	≒	≒	≒	/fallingdotseq R: eq, falling dots
U+22DD	⪖	□	□	□	□	/eqslantgtr R: equal-or-gtr, slanted
U+22DC	⪕	□	□	□	□	/eqslantless R: eq-or-less, slanted
U+2253	≓	≓	≓	≓	≓	/risingdotseq R: eq, rising dots
U+22D4	⋔	⋔	⋔	⋔	⋔	/pitchfork R: pitchfork
U+2322	⌢	□	□	□	□	/frown R: down curve
U+2273	⪆	≳	≳	≳	≳	/gtrapprox R: greater, approximate
U+22D7	&gsdot;	⋗	⋗	⋗	⋗	/gtrdot R: greater than, with dot
U+2267	≧	≧	≧	≧	≧	/geqq R: greater, double equals
U+22DB	⋛	⋛	⋛	⋛	⋛	/gtreqless R: greater, equals, less
U+22DB	⪌	⋛	⋛	⋛	⋛	/gtreqqless R: gt, dbl equals, less
U+2273	⩾	≥	≥	≥	≥	/geqslant R: gt-or-equal, slanted
U+22D9	⋙	⋙	⋙	⋙	⋙	/ggg /Gg /gggtr R: triple gtr-than
U+2277	≷	≷	≷	≷	≷	/gtrless R: greater, less
U+2273	≳	≳	≳	≳	≳	/gtrsim R: greater, similar
U+226B	≫	≫	≫	≫	≫	/gg R: dbl greater-than sign
U+2272	⪅	≲	≲	≲	≲	/lessapprox R: less, approximate
U+22D6	&ldot;	⋖	⋖	⋖	⋖	/lessdot R: less than, with dot
U+2266	≦	≦	≦	≦	≦	/leqq R: less, double equals
U+22DA	⪋	⋚	⋚	⋚	⋚	/lesseqqgtr R: less, dbl eq, greater
U+22DA	⋚	⋚	⋚	⋚	⋚	/lesseqgtr R: less, eq, greater
U+2264	⩽	≤	≤	≤	≤	/leqslant R: less-than-or-eq, slant
U+2276	≶	≶	≶	≶	≶	/lessgtr R: less, greater
U+22D8	⋘	⋘	⋘	⋘	⋘	/Ll /lll /llless R: triple less-than
U+2272	≲	≲	≲	≲	≲	/lesssim R: less, similar
U+226A	≪	≪	≪	≪	≪	/ll R: double less-than sign
U+22B4	⊴	⊴	⊴	⊴	⊴	/trianglelefteq R: left triangle, eq
U+2223	∣	∣	∣	∣	∣	/mid R:
U+22A7	⊧	□	□	□	□	/models R:
U+227A	≺	≺	≺	≺	≺	/prec R: precedes
U+227E	⪷	≾	≾	≾	≾	/precapprox R: precedes, approximate
U+227C	⪯	≼	≼	≼	≼	/preceq R: precedes, equals
U+227E	≾	≾	≾	≾	≾	/precsim R: precedes, similar
U+22B5	⊵	⊵	⊵	⊵	⊵	/trianglerighteq R: right tri, eq
U+2210	&samalg;	∐	∐	∐	∐	/smallamalg R: small amalg
U+227B	≻	≻	≻	≻	≻	/succ R: succeeds
U+227F	⪸	≿	≿	≿	≿	/succapprox R: succeeds, approximate
U+227D	≽	≽	≽	≽	≽	/succcurlyeq R: succeeds, curly eq
U+227D	⪰	≽	≽	≽	≽	/succeq R: succeeds, equals
U+227F	≿	≿	≿	≿	≿	/succsim R: succeeds, similar
U+2322	⌢	□	□	□	□	/smallfrown R: small down curve
U+E301	∣	□	□	□	□	/shortmid R:
U+2323	⌣	□	□	□	□	/smile R: up curve
U+2225	∥	∥	∥	∥	∥	/shortparallel R: short parallel
U+228F	⊏	⊏	⊏	⊏	⊏	/sqsubset R: square subset
U+2291	⊑	⊑	⊑	⊑	⊑	/sqsubseteq R: square subset, equals
U+2290	⊐	⊐	⊐	⊐	⊐	/sqsupset R: square superset
U+2292	⊒	⊒	⊒	⊒	⊒	/sqsupseteq R: square superset, eq
U+2323	⌣	□	□	□	□	/smallsmile R: small up curve
U+22D0	⋐	⋐	⋐	⋐	⋐	/Subset R: double subset
U+2286	⫅	⊆	⊆	⊆	⊆	/subseteqq R: subset, dbl equals
U+22D1	⋑	⋑	⋑	⋑	⋑	/Supset R: dbl superset
U+2287	⫆	⊇	⊇	⊇	⊇	/supseteqq R: superset, dbl equals
U+2248	≈	≈	≈	≈	≈	/thickapprox R: thick approximate
U+223C	∼	~	~	~	~	/thicksim R: thick similar
U+225C	≜	≜	≜	≜	≜	/triangleq R: triangle, equals
U+226C	≬	≬	≬	≬	≬	/between R: between
U+22A2	⊢	⊢	⊢	⊢	⊢	/vdash R: vertical, dash
U+22A9	⊩	⊩	⊩	⊩	⊩	/Vdash R: dbl vertical, dash
U+22A8	⊨	□	□	□	□	/vDash R: vertical, dbl dash
U+22BB	⊻	⊻	⊻	⊻	⊻	/veebar B: logical or, bar below
U+22B2	⊲	◁	◁	◁	◁	/vartriangleleft R: l tri, open, var
U+221D	∝	∝	∝	∝	∝	/varpropto R: proportional, variant
U+22B3	⊳	▷	▷	▷	▷	/vartriangleright R: r tri, open, var
U+22AA	⊪	⊪	⊪	⊪	⊪	/Vvdash R: triple vertical, dash

Tabella u79.10. Entità ISO 8879:1986 ISOamsn: *added math symbols: negated relations.*

Punto di codifica	Macro SGML					Descrizione
U+E411	⪊	□	□	□	□	/gnapprox N: greater, not approximate
U+2269	⪈	≩	≩	≩	≩	/gneq N: greater, not equals
U+2269	≩	≩	≩	≩	≩	/gneqq N: greater, not dbl equals
U+22E7	⋧	≩	≩	≩	≩	/gnsim N: greater, not similar
U+2269	≩︀	≩	≩	≩	≩	/gvertneqq N: gt, vert, not dbl eq
U+E2A2	⪉	□	□	□	□	/lnapprox N: less, not approximate

Punto di codifica	Macro SGML					Descrizione
U+2268	≨	≨	≨	≨	≨	/lneqq N: less, not double equals
U+2268	⪇	≨	≨	≨	≨	/lneq N: less, not equals
U+22E6	⋦	≨	≨	≨	≨	/lnsim N: less, not similar
U+2268	≨︀	≨	≨	≨	≨	/lvertneqq N: less, vert, not dbl eq
U+2249	≉	≉	≉	≉	≉	/napprox N: not approximate
U+2247	≇	≇	≇	≇	≇	/ncong N: not congruent with
U+2262	≢	≢	≢	≢	≢	/nequiv N: not identical with
U+2271	≧̸	≱	≱	≱	≱	/ngeqq N: not greater, dbl equals
U+2271	≱	≱	≱	≱	≱	/ngeq N: not greater-than-or-equal
U+2271	⩾̸	≱	≱	≱	≱	/ngeqslant N: not gt-or-eq, slanted
U+226F	≯	≯	≯	≯	≯	/ngtr N: not greater-than
U+2270	≰	≰	≰	≰	≰	/nleq N: not less-than-or-equal
U+2270	≦̸	≰	≰	≰	≰	/nleqq N: not less, dbl equals
U+2270	⩽̸	≰	≰	≰	≰	/nleqslant N: not less-or-eq, slant
U+226E	≮	≮	≮	≮	≮	/nless N: not less-than
U+22EA	⋪	⋪	⋪	⋪	⋪	/ntriangleleft N: not left triangle
U+22EC	⋬	⋬	⋬	⋬	⋬	/ntrianglelefteq N: not l tri, eq
U+2224	∤	∤	∤	∤	∤	/nmid
U+2226	∦	∦	∦	∦	∦	/nparallel N: not parallel
U+2280	⊀	⊀	⊀	⊀	⊀	/nprec N: not precedes
U+22E0	⪯̸	⋠	⋠	⋠	⋠	/npreceq N: not precedes, equals
U+22EB	⋫	⋫	⋫	⋫	⋫	/ntriangleright N: not rt triangle
U+22ED	⋭	⋭	⋭	⋭	⋭	/ntrianglerighteq N: not r tri, eq
U+2281	⊁	⊁	⊁	⊁	⊁	/nsucc N: not succeeds
U+22E1	⪰̸	⋡	⋡	⋡	⋡	/nsucceq N: not succeeds, equals
U+2241	≁	≁	≁	≁	≁	/nsim N: not similar
U+2244	≄	≄	≄	≄	≄	/nsimeq N: not similar, equals
U+E2AA	∤	□	□	□	□	/nshortmid
U+2226	∦	∦	∦	∦	∦	/nshortparallel N: not short par
U+2284	⊄	⊄	⊄	⊄	⊄	not a subset of
U+2288	⊈	⊈	⊈	⊈	⊈	/nsubseteq N: not subset, equals
U+2288	⫅̸	⊈	⊈	⊈	⊈	/nsubseteqq N: not subset, dbl eq
U+2285	⊅	⊅	⊅	⊅	⊅	not superset
U+2289	⫆̸	⊉	⊉	⊉	⊉	/nsupseteqq N: not superset, dbl eq
U+2289	⊉	⊉	⊉	⊉	⊉	/nsupseteq N: not superset, equals
U+22AC	⊬	□	□	□	□	/nvdash N: not vertical, dash
U+22AD	⊭	□	□	□	□	/nvDash N: not vertical, dbl dash
U+22AF	⊯	□	□	□	□	/nVDash N: not dbl vert, dbl dash
U+22AE	⊮	⊮	⊮	⊮	⊮	/nVdash N: not dbl vertical, dash
U+22E8	⪹	⋨	⋨	⋨	⋨	/precnapprox N: precedes, not approx
U+E2B3	⪵	□	□	□	□	/precneqq N: precedes, not dbl eq
U+22E8	⋨	⋨	⋨	⋨	⋨	/precnsim N: precedes, not similar
U+22E9	⪺	⋩	⋩	⋩	⋩	/succnapprox N: succeeds, not approx
U+E2B5	⪶	□	□	□	□	/succneqq N: succeeds, not dbl eq
U+22E9	⋩	⋩	⋩	⋩	⋩	/succnsim N: succeeds, not similar
U+228A	⊊	⊊	⊊	⊊	⊊	/subsetneq N: subset, not equals
U+228A	⫋	⊊	⊊	⊊	⊊	/subsetneqq N: subset, not dbl eq
U+228B	⊋	⊋	⊋	⊋	⊋	/supsetneq N: superset, not equals
U+228B	⫌	⊋	⊋	⊋	⊋	/supsetneqq N: superset, not dbl eq
U+E2B8	⫋︀	□	□	□	□	/varsubsetneqq N: subset not dbl eq, var
U+228A	⊊︀	⊊	⊊	⊊	⊊	/varsubsetneq N: subset, not eq, var
U+228B	⊋︀	⊋	⊋	⊋	⊋	/varsupsetneq N: superset, not eq, var
U+228E	⫌︀	⊋	⊋	⊋	⊋	/varsupsetneqq N: super not dbl eq, var

Tabella u79.11. Entità ISO 8879:1986 ISOamsa: *added math symbols: arrow relations.*

Punto di codifica	Macro SGML					Descrizione
U+21B6	↶	□	□	□	□	/curvearrowleft A: left curved arrow
U+21B7	↷	□	□	□	□	/curvearrowright A: rt curved arrow
U+21D3	⇓	⇓	⇓	⇓	⇓	downwards double arrow
U+21CA	&darr2;	□	□	□	□	/downdownarrows A: two down arrows
U+21C3	⇃	□	□	□	□	/downharpoonleft A: dn harpoon-left
U+21C2	⇂	□	□	□	□	/downharpoonright A: down harpoon-rt
U+21DA	⇚	□	□	□	□	/Lleftarrow A: left triple arrow
U+219E	↞	□	□	□	□	/twoheadleftarrow A:
U+21C7	&larr2;	□	□	□	□	/leftleftarrows A: two left arrows
U+21A9	↩	□	□	□	□	/hookleftarrow A: left arrow-hooked
U+21AB	↫	□	□	□	□	/looparrowleft A: left arrow-looped
U+21A2	↢	□	□	□	□	/leftarrowtail A: left arrow-tailed
U+21BD	↽	□	□	□	□	/leftharpoondown A: l harpoon-down
U+21BC	↼	□	□	□	□	/leftharpoonup A: left harpoon-up
U+21D4	⇔	⇔	⇔	⇔	⇔	left right double arrow
U+2194	↔	↔	↔	↔	↔	left right arrow
U+21C6	&lrarr2;	□	□	□	□	/leftrightarrows A: l arr over r arr
U+21C4	&rlarr2;	□	□	□	□	/rightleftarrows A: r arr over l arr
U+21AD	↭	□	□	□	□	/leftrightsquigarrow A: l&r arr-wavy

Punto di codifica	Macro SGML					Descrizione
U+21CC	&rlhar2;	⇌	⇌	⇌	⇌	/rightleftharpoons A: r harp over l
U+21CB	&lrhar2;	□	□	□	□	/leftrightharpoons A: l harp over r
U+21B0	↰	□	□	□	□	/Lsh A:
U+21A6	↦	□	□	□	□	/mapsto A:
U+22B8	⊸	⊸	⊸	⊸	⊸	/multimap A:
U+2197	↗	□	□	□	□	/nearrow A: NE pointing arrow
U+21CD	⇍	□	□	□	□	/nLeftarrow A: not implied by
U+219A	↚	□	□	□	□	/nleftarrow A: not left arrow
U+21CE	⇎	□	□	□	□	/nLeftrightarrow A: not l&r dbl arr
U+21AE	↮	□	□	□	□	/nleftrightarrow A: not l&r arrow
U+219B	↛	□	□	□	□	/nrightarrow A: not right arrow
U+21CF	⇏	□	□	□	□	/nRightarrow A: not implies
U+2196	↖	□	□	□	□	/nwarrow A: NW pointing arrow
U+21BA	↺	□	□	□	□	/circlearrowleft A: l arr in circle
U+21BB	↻	□	□	□	□	/circlearrowright A: r arr in circle
U+21DB	&rrAarr;	□	□	□	□	/Rrightarrow A: right triple arrow
U+21A0	↠	□	□	□	□	/twoheadrightarrow A:
U+21C9	&rarr2;	□	□	□	□	/rightrightarrows A: two rt arrows
U+21AA	↪	□	□	□	□	/hookrightarrow A: rt arrow-hooked
U+21AC	↬	□	□	□	□	/looparrowright A: rt arrow-looped
U+21A3	↣	□	□	□	□	/rightarrowtail A: rt arrow-tailed
U+219D	↝	⇝	⇝	⇝	⇝	/rightsquigarrow A: rt arrow-wavy
U+21C1	⇁	□	□	□	□	/rightharpoondown A: rt harpoon-down
U+21C0	⇀	□	□	□	□	/rightharpoonup A: rt harpoon-up
U+21B1	↱	□	□	□	□	/Rsh A:
U+2198	&drarr;	□	□	□	□	/searrow A: SE pointing arrow
U+2199	&dlarr;	□	□	□	□	/swarrow A: SW pointing arrow
U+21D1	⇑	⇑	⇑	⇑	⇑	upwards double arrow
U+21C8	&uarr2;	□	□	□	□	/upuparrows A: two up arrows
U+21D5	⇕	□	□	□	□	/Updownarrow A: up&down dbl arrow
U+2195	↕	↕	↕	↕	↕	/updownarrow A: up&down arrow
U+21BF	↿	□	□	□	□	/upharpoonleft A: up harpoon-left
U+21BE	↾	□	□	□	□	/upharpoonright /restriction A: up harpr
U+21D0	⟸	⇐	⇐	⇐	⇐	/Longleftarrow A: long l dbl arrow
U+2194	⟺	↔	↔	↔	↔	/Longleftrightarrow A: long l&r dbl arr
U+2194	⟷	↔	↔	↔	↔	/longleftrightarrow A: long l&r arr
U+21D2	⟹	⇒	⇒	⇒	⇒	/Longrightarrow A: long rt dbl arr

Tabella u79.12. Entità ISO 8879:1986 ISOamsc: *added math symbols: delimiters.*

Punto di codifica	Macro SGML					Descrizione
U+2309	⌉	⌉	⌉	⌉	⌉	right ceiling
U+230B	⌋	⌋	⌋	⌋	⌋	right floor
U+E291	⦔	□	□	□	□	C: right paren, gt
U+231D	⌝	□	□	□	□	/urcorner C: upper right corner
U+231F	⌟	□	□	□	□	/lrcorner C: lower right corner
U+2308	⌈	⌈	⌈	⌈	⌈	left ceiling = apl upstile
U+230A	⌊	⌊	⌊	⌊	⌊	left floor = apl downstile
	&lpargt;	⊁	⊁	⊁	⊁	/leftparengtr O: left parenthesis, gt
U+231C	⌜	□	□	□	□	/ulcorner O: upper left corner
U+231E	⌞	□	□	□	□	/llcorner O: lower left corner

Alfabeti latini

Tabella u79.13. Entità ISO 8879:1986 ISOlat1: *added latin 1.*

Punto di codifica	Macro SGML					Descrizione
U+00E1	á	á	*á*	**á**	**á**	latin small letter a with acute
U+00C1	Á	Á	*Á*	**Á**	**Á**	latin capital letter A with acute
U+00E2	â	â	*â*	**â**	**â**	latin capital letter A with circumflex
U+00C2	Â	Â	*Â*	**Â**	**Â**	latin small letter a with circumflex
U+00E0	à	à	*à*	**à**	**à**	latin small letter a with grave = latin small letter a grave
U+00C0	À	À	*À*	**À**	**À**	latin capital letter A with grave = latin capital letter A grave
U+00E5	å	å	*å*	**å**	**å**	latin small letter a with ring above = latin small letter a ring
U+00C5	Å	Å	*Å*	**Å**	**Å**	latin capital letter A with ring above = latin capital letter A ring
U+00E3	ã	ã	*ã*	**ã**	**ã**	latin small letter a with tilde
U+00C3	Ã	Ã	*Ã*	**Ã**	**Ã**	latin capital letter A with tilde
U+00E4	ä	ä	*ä*	**ä**	**ä**	latin small letter a with diaeresis
U+00C4	Ä	Ä	*Ä*	**Ä**	**Ä**	latin capital letter A with diaeresis
U+00E6	æ	æ	*æ*	**æ**	**æ**	latin small letter ae = latin small ligature ae
U+00C6	Æ	Æ	*Æ*	**Æ**	**Æ**	latin capital letter AE = latin capital ligature AE

Punto di codifica	Macro SGML					Descrizione
U+00E7	ç	ç	*ç*	**ç**	**ç**	latin small letter c with cedilla
U+00C7	Ç	Ç	*Ç*	**Ç**	**Ç**	latin capital letter C with cedilla
U+00F0	ð	ð	*ð*	**ð**	**ð**	latin small letter eth
U+00D0	Ð	Ð	*Ð*	**Ð**	**Ð**	latin capital letter ETH
U+00E9	é	é	*é*	**é**	**é**	latin small letter e with acute
U+00C9	É	É	*É*	**É**	**É**	latin capital letter E with acute
U+00EA	ê	ê	*ê*	**ê**	**ê**	latin small letter e with circumflex
U+00CA	Ê	Ê	*Ê*	**Ê**	**Ê**	latin capital letter E with circumflex
U+00E8	è	è	*è*	**è**	**è**	latin small letter e with grave
U+00C8	È	È	*È*	**È**	**È**	latin capital letter E with grave
U+00EB	ë	ë	*ë*	**ë**	**ë**	latin small letter e with diaeresis
U+00CB	Ë	Ë	*Ë*	**Ë**	**Ë**	latin capital letter E with diaeresis
U+00ED	í	í	*í*	**í**	**í**	latin small letter i with acute
U+00CD	Í	Í	*Í*	**Í**	**Í**	latin capital letter I with acute
U+00EE	î	î	*î*	**î**	**î**	latin small letter i with circumflex
U+00CE	Î	Î	*Î*	**Î**	**Î**	latin capital letter I with circumflex
U+00EC	ì	ì	*ì*	**ì**	**ì**	latin small letter i with grave
U+00CC	Ì	Ì	*Ì*	**Ì**	**Ì**	latin capital letter I with grave
U+00EF	ï	ï	*ï*	**ï**	**ï**	latin small letter i with diaeresis
U+00CF	Ï	Ï	*Ï*	**Ï**	**Ï**	latin capital letter I with diaeresis
U+00F1	ñ	ñ	*ñ*	**ñ**	**ñ**	latin small letter n with tilde
U+00D1	Ñ	Ñ	*Ñ*	**Ñ**	**Ñ**	latin capital letter N with tilde
U+00F3	ó	ó	*ó*	**ó**	**ó**	latin small letter o with acute
U+00D3	Ó	Ó	*Ó*	**Ó**	**Ó**	latin capital letter O with acute
U+00F4	ô	ô	*ô*	**ô**	**ô**	latin small letter o with circumflex
U+00D4	Ô	Ô	*Ô*	**Ô**	**Ô**	latin capital letter O with circumflex
U+00F2	ò	ò	*ò*	**ò**	**ò**	latin small letter o with grave
U+00D2	Ò	Ò	*Ò*	**Ò**	**Ò**	latin capital letter O with grave
U+00F8	ø	ø	*ø*	**ø**	**ø**	latin capital letter O with stroke = latin capital letter O slash
U+00D8	Ø	Ø	*Ø*	**Ø**	**Ø**	latin capital letter O with stroke = latin capital letter O slash
U+00F5	õ	õ	*õ*	**õ**	**õ**	latin small letter o with tilde
U+00D5	Õ	Õ	*Õ*	**Õ**	**Õ**	latin capital letter O with tilde
U+00F6	ö	ö	*ö*	**ö**	**ö**	latin small letter o with diaeresis
U+00D6	Ö	Ö	*Ö*	**Ö**	**Ö**	latin capital letter O with diaeresis
U+00DF	ß	ß	*ß*	**ß**	**ß**	latin small letter sharp s = ess-zed
U+00FE	þ	þ	*þ*	**þ**	**þ**	latin small letter thorn with
U+00DE	Þ	Þ	*Þ*	**Þ**	**Þ**	latin capital letter THORN
U+00FA	ú	ú	*ú*	**ú**	**ú**	latin small letter u with acute
U+00DA	Ú	Ú	*Ú*	**Ú**	**Ú**	latin capital letter U with acute
U+00FB	û	û	*û*	**û**	**û**	latin small letter u with circumflex
U+00DB	Û	Û	*Û*	**Û**	**Û**	latin capital letter U with circumflex
U+00F9	ù	ù	*ù*	**ù**	**ù**	latin small letter u with grave
U+00D9	Ù	Ù	*Ù*	**Ù**	**Ù**	latin capital letter U with grave
U+00FC	ü	ü	*ü*	**ü**	**ü**	latin small letter u with diaeresis
U+00DC	Ü	Ü	*Ü*	**Ü**	**Ü**	latin capital letter U with diaeresis
U+00FD	ý	ý	*ý*	**ý**	**ý**	latin small letter y with acute
U+00DD	Ý	Ý	*Ý*	**Ý**	**Ý**	latin capital letter Y with acute
U+00FF	ÿ	ÿ	*ÿ*	**ÿ**	**ÿ**	latin small letter y with diaeresis

Tabella u79.14. Entità ISO 8879:1986 ISOlat2: *added latin 2.*

Punto di codifica	Macro SGML					Descrizione
U+0103	ă	ă	*ă*	**ă**	**ă**	=small a, breve
U+0102	Ă	Ă	*Ă*	**Ă**	**Ă**	=capital A, breve
U+0101	ā	ā	*ā*	**ā**	**ā**	=small a, macron
U+0100	Ā	Ā	*Ā*	**Ā**	**Ā**	=capital A, macron
U+0105	ą	ą	*ą*	**ą**	**ą**	=small a, ogonek
U+0104	Ą	Ą	*Ą*	**Ą**	**Ą**	=capital A, ogonek
U+0107	ć	ć	*ć*	**ć**	**ć**	=small c, acute accent
U+0106	Ć	Ć	*Ć*	**Ć**	**Ć**	=capital C, acute accent
U+010D	č	č	*č*	**č**	**č**	=small c, caron
U+010C	Č	Č	*Č*	**Č**	**Č**	=capital C, caron
U+0109	ĉ	ĉ	*ĉ*	**ĉ**	**ĉ**	=small c, circumflex accent
U+0108	Ĉ	Ĉ	*Ĉ*	**Ĉ**	**Ĉ**	=capital C, circumflex accent
U+010B	ċ	ċ	*ċ*	**ċ**	**ċ**	=small c, dot above
U+010A	Ċ	Ċ	*Ċ*	**Ċ**	**Ċ**	=capital C, dot above
U+010F	ď	ď	*ď*	**ď**	**ď**	=small d, caron
U+010E	Ď	Ď	*Ď*	**Ď**	**Ď**	=capital D, caron
U+0111	đ	đ	*đ*	**đ**	**đ**	=small d, stroke
U+0110	Đ	Đ	*Đ*	**Đ**	**Đ**	=capital D, stroke
U+011B	ě	ě	*ě*	**ě**	**ě**	=small e, caron
U+011A	Ě	Ě	*Ě*	**Ě**	**Ě**	=capital E, caron
U+0117	ė	ė	*ė*	**ė**	**ė**	=small e, dot above
U+0116	Ė	Ė	*Ė*	**Ė**	**Ė**	=capital E, dot above
U+0113	ē	ē	*ē*	**ē**	**ē**	=small e, macron
U+0112	Ē	Ē	*Ē*	**Ē**	**Ē**	=capital E, macron

Punto di codifica	Macro SGML					Descrizione
U+0119	ę	ę	*ę*	**ę**	***ę***	=small e, ogonek
U+0118	Ę	Ę	*Ę*	**Ę**	***Ę***	=capital E, ogonek
U+01F5	ǵ	ǵ	*ǵ*	**ǵ**	***ǵ***	=small g, acute accent
U+011F	ğ	ğ	*ğ*	**ğ**	***ğ***	=small g, breve
U+011E	Ğ	Ğ	*Ğ*	**Ğ**	***Ğ***	=capital G, breve
U+0122	Ģ	Ģ	*Ģ*	**Ģ**	***Ģ***	=capital G, cedilla
U+011D	ĝ	ĝ	*ĝ*	**ĝ**	***ĝ***	=small g, circumflex accent
U+011C	Ĝ	Ĝ	*Ĝ*	**Ĝ**	***Ĝ***	=capital G, circumflex accent
U+0121	ġ	ġ	*ġ*	**ġ**	***ġ***	=small g, dot above
U+0120	Ġ	Ġ	*Ġ*	**Ġ**	***Ġ***	=capital G, dot above
U+0125	ĥ	ĥ	*ĥ*	**ĥ**	***ĥ***	=small h, circumflex accent
U+0124	Ĥ	Ĥ	*Ĥ*	**Ĥ**	***Ĥ***	=capital H, circumflex accent
U+0127	ħ	ħ	*ħ*	**ħ**	***ħ***	=small h, stroke
U+0126	Ħ	Ħ	*Ħ*	**Ħ**	***Ħ***	=capital H, stroke
U+0130	İ	İ	*İ*	**İ**	***İ***	=capital I, dot above
U+012A	Ī	Ī	*Ī*	**Ī**	***Ī***	=capital I, macron
U+012B	ī	ī	*ī*	**ī**	***ī***	=small i, macron
U+0133	ĳ	ij	*ij*	**ij**	***ij***	=small ij ligature
U+0132	Ĳ	IJ	*IJ*	**IJ**	***IJ***	=capital IJ ligature
U+0131	ı	ı	*ı*	**ı**	***ı***	=small i without dot
U+012F	į	į	*į*	**į**	***į***	=small i, ogonek
U+012E	Į	Į	*Į*	**Į**	***Į***	=capital I, ogonek
U+0129	ĩ	ĩ	*ĩ*	**ĩ**	***ĩ***	=small i, tilde
U+0128	Ĩ	Ĩ	*Ĩ*	**Ĩ**	***Ĩ***	=capital I, tilde
U+0135	ĵ	ĵ	*ĵ*	**ĵ**	***ĵ***	=small j, circumflex accent
U+0134	Ĵ	Ĵ	*Ĵ*	**Ĵ**	***Ĵ***	=capital J, circumflex accent
U+0137	ķ	ķ	*ķ*	**ķ**	***ķ***	=small k, cedilla
U+0136	Ķ	Ķ	*Ķ*	**Ķ**	***Ķ***	=capital K, cedilla
U+0138	ĸ	□	□	□	□	=small k, Greenlandic
U+013A	ĺ	ĺ	*ĺ*	**ĺ**	***ĺ***	=small l, acute accent
U+0139	Ĺ	Ĺ	*Ĺ*	**Ĺ**	***Ĺ***	=capital L, acute accent
U+013E	ľ	ľ	*ľ*	**ľ**	***l′***	=small l, caron
U+013D	Ľ	Ľ	*Ľ*	**Ľ**	***L′***	=capital L, caron
U+013C	ļ	ļ	*ļ*	**ļ**	***ļ***	=small l, cedilla
U+013B	Ļ	Ļ	*Ļ*	**Ļ**	***Ļ***	=capital L, cedilla
U+0140	ŀ	l·	*l·*	**l·**	***l·***	=small l, middle dot
U+013F	Ŀ	L·	*L·*	**L·**	***L·***	=capital L, middle dot
U+0142	ł	ł	*ł*	**ł**	***ł***	=small l, stroke
U+0141	Ł	Ł	*Ł*	**Ł**	***Ł***	=capital L, stroke
U+0144	ń	ń	*ń*	**ń**	***ń***	=small n, acute accent
U+0143	Ń	Ń	*Ń*	**Ń**	***Ń***	=capital N, acute accent
U+014B	ŋ	■	■	■	■	=small eng, Lapp
U+014A	Ŋ	■	■	■	■	=capital ENG, Lapp
U+0149	ŉ	□	□	□	□	=small n, apostrophe
U+0148	ň	ň	*ň*	**ň**	***ň***	=small n, caron
U+0147	Ň	Ň	*Ň*	**Ň**	***Ň***	=capital N, caron
U+0146	ņ	ņ	*ņ*	**ņ**	***ņ***	=small n, cedilla
U+0145	Ņ	Ņ	*Ņ*	**Ņ**	***Ņ***	=capital N, cedilla
U+0151	ő	ő	*ő*	**ő**	***ő***	=small o, double acute accent
U+0150	Ő	Ő	*Ő*	**Ő**	***Ő***	=capital O, double acute accent
U+014C	Ō	Ō	*Ō*	**Ō**	***Ō***	=capital O, macron
U+014D	ō	ō	*ō*	**ō**	***ō***	=small o, macron
U+0153	œ	œ	*œ*	**œ**	***œ***	latin small ligature oe
U+0152	Œ	Œ	*Œ*	**Œ**	***Œ***	latin capital ligature OE
U+0155	ŕ	ŕ	*ŕ*	**ŕ**	***ŕ***	=small r, acute accent
U+0154	Ŕ	Ŕ	*Ŕ*	**Ŕ**	***Ŕ***	=capital R, acute accent
U+0159	ř	ř	*ř*	**ř**	***ř***	=small r, caron
U+0158	Ř	Ř	*Ř*	**Ř**	***Ř***	=capital R, caron
U+0157	ŗ	ŗ	*ŗ*	**ŗ**	***ŗ***	=small r, cedilla
U+0156	Ŗ	Ŗ	*Ŗ*	**Ŗ**	***Ŗ***	=capital R, cedilla
U+015B	ś	ś	*ś*	**ś**	***ś***	=small s, acute accent
U+015A	Ś	Ś	*Ś*	**Ś**	***Ś***	=capital S, acute accent
U+0161	š	š	*š*	**š**	***š***	latin small letter s with caron
U+0160	Š	Š	*Š*	**Š**	***Š***	latin capital letter S with caron
U+015F	ş	ş	*ş*	**ş**	***ş***	=small s, cedilla
U+015E	Ş	Ş	*Ş*	**Ş**	***Ş***	=capital S, cedilla
U+015D	ŝ	ŝ	*ŝ*	**ŝ**	***ŝ***	=small s, circumflex accent
U+015C	Ŝ	Ŝ	*Ŝ*	**Ŝ**	***Ŝ***	=capital S, circumflex accent
U+0165	ť	ť	*ť*	**ť**	***t′***	=small t, caron
U+0164	Ť	Ť	*Ť*	**Ť**	***T′***	=capital T, caron
U+0163	ţ	ţ	*ţ*	**ţ**	***ţ***	=small t, cedilla
U+0162	Ţ	Ţ	*Ţ*	**Ţ**	***Ţ***	=capital T, cedilla
U+0167	ŧ	ŧ	*ŧ*	**ŧ**	***ŧ***	=small t, stroke
U+0166	Ŧ	Ŧ	*Ŧ*	**Ŧ**	***Ŧ***	=capital T, stroke
U+016D	ŭ	ŭ	*ŭ*	**ŭ**	***ŭ***	=small u, breve
U+016C	Ŭ	Ŭ	*Ŭ*	**Ŭ**	***Ŭ***	=capital U, breve
U+0171	ű	ű	*ű*	**ű**	***ű***	=small u, double acute accent

Punto di codifica	Macro SGML					Descrizione
U+0170	Ű	Ű	*Ű*	**Ű**	***Ű***	=capital U, double acute accent
U+016B	ū	ū	*ū*	**ū**	***ū***	=small u, macron
U+016A	Ū	Ū	*Ū*	**Ū**	***Ū***	=capital U, macron
U+0173	ų	ų	*ų*	**ų**	***ų***	=small u, ogonek
U+0172	Ų	Ų	*Ų*	**Ų**	***Ų***	=capital U, ogonek
U+016F	ů	ů	*ů*	**ů**	***ů***	=small u, ring
U+016E	Ů	Ů	*Ů*	**Ů**	***Ů***	=capital U, ring
U+0169	ũ	ũ	*ũ*	**ũ**	***ũ***	=small u, tilde
U+0168	Ũ	Ũ	*Ũ*	**Ũ**	***Ũ***	=capital U, tilde
U+0175	ŵ	ŵ	*ŵ*	**ŵ**	***ŵ***	=small w, circumflex accent
U+0174	Ŵ	Ŵ	*Ŵ*	**Ŵ**	***Ŵ***	=capital W, circumflex accent
U+0177	ŷ	ŷ	*ŷ*	**ŷ**	***ŷ***	=small y, circumflex accent
U+0176	Ŷ	Ŷ	*Ŷ*	**Ŷ**	***Ŷ***	=capital Y, circumflex accent
U+0178	Ÿ	Ÿ	*Ÿ*	**Ÿ**	***Ÿ***	latin capital letter Y with diaeresis
U+017A	ź	ź	*ź*	**ź**	***ź***	=small z, acute accent
U+0179	Ź	Ź	*Ź*	**Ź**	***Ź***	=capital Z, acute accent
U+017E	ž	ž	*ž*	**ž**	***ž***	=small z, caron
U+017D	Ž	Ž	*Ž*	**Ž**	***Ž***	=capital Z, caron
U+017C	ż	ż	*ż*	**ż**	***ż***	=small z, dot above
U+017B	Ż	Ż	*Ż*	**Ż**	***Ż***	=capital Z, dot above

Tabella u79.15. Entità ISO 8879:1986 ISOdia: *diacritical marks*.

Punto di codifica	Macro SGML					Descrizione
U+00B4	´	´	´	´	´	acute accent = spacing acute
U+02D8	˘	˘	˘	˘	˘	=breve
U+02C7	ˇ	ˇ	ˇ	ˇ	ˇ	=caron
U+00B8	¸	¸	¸	¸	¸	cedilla = spacing cedilla
U+005E	ˆ	^	^	^	^	modifier letter circumflex accent
U+02DD	˝	˝	˝	˝	˝	=double acute accent
U+00A8	¨	¨	¨	¨	¨	diaeresis = spacing diaeresis
U+02D9	˙	˙	˙	˙	˙	=dot above
U+0060	`	`	`	`	`	=grave accent
U+00AF	¯	¯	¯	¯	¯	macron = spacing macron = overline = APL overbar
U+02DB	˛	˛	˛	˛	˛	=ogonek
U+02DA	˚	˚	˚	˚	˚	=ring
U+02DC	˜	~	~	~	~	small tilde
U+00A8	¨	¨	¨	¨	¨	diaeresis = spacing diaeresis

Alfabeti non latini

«

Si ricorda che per poter utilizzare gli alfabeti non latini è indispensabile selezionare il linguaggio.

Si seleziona il russo con la sigla '**ru**'.

Tabella u79.16. Entità ISO 8879:1986 ISOcyr1: *russian cyrillic*.

Punto di codifica	Macro SGML					Descrizione
U+0430	а	а	*а*	**а**	***а***	=small a, Cyrillic
U+0410	А	А	*А*	**А**	***А***	=capital A, Cyrillic
U+0431	б	б	*б*	**б**	***б***	=small be, Cyrillic
U+0411	Б	Б	*Б*	**Б**	***Б***	=capital BE, Cyrillic
U+0432	в	в	*в*	**в**	***в***	=small ve, Cyrillic
U+0412	В	В	*В*	**В**	***В***	=capital VE, Cyrillic
U+0433	г	г	*г*	**г**	***г***	=small ghe, Cyrillic
U+0413	Г	Г	*Г*	**Г**	***Г***	=capital GHE, Cyrillic
U+0434	д	д	*д*	**д**	***д***	=small de, Cyrillic
U+0414	Д	Д	*Д*	**Д**	***Д***	=capital DE, Cyrillic
U+0435	е	е	*е*	**е**	***е***	=small ie, Cyrillic
U+0415	Е	Е	*Е*	**Е**	***Е***	=capital IE, Cyrillic
U+0451	ё	ё	*ё*	**ё**	***ё***	=small io, Russian
U+0401	Ё	Ё	*Ё*	**Ё**	***Ё***	=capital IO, Russian
U+0436	ж	ж	*ж*	**ж**	***ж***	=small zhe, Cyrillic
U+0416	Ж	Ж	*Ж*	**Ж**	***Ж***	=capital ZHE, Cyrillic
U+0437	з	з	*з*	**з**	***з***	=small ze, Cyrillic
U+0417	З	З	*З*	**З**	***З***	=capital ZE, Cyrillic
U+0438	и	и	*и*	**и**	***и***	=small i, Cyrillic
U+0418	И	И	*И*	**И**	***И***	=capital I, Cyrillic
U+0439	й	й	*й*	**й**	***й***	=small short i, Cyrillic
U+0419	Й	Й	*Й*	**Й**	***Й***	=capital short I, Cyrillic
U+043A	к	к	*к*	**к**	***к***	=small ka, Cyrillic
U+041A	К	К	*К*	**К**	***К***	=capital KA, Cyrillic
U+043B	л	л	*л*	**л**	***л***	=small el, Cyrillic
U+041B	Л	Л	*Л*	**Л**	***Л***	=capital EL, Cyrillic
U+043C	м	м	*м*	**м**	***м***	=small em, Cyrillic

Punto di codifica	Macro SGML					Descrizione
U+041C	М	М	М	М	М	=capital EM, Cyrillic
U+043D	н	н	н	н	н	=small en, Cyrillic
U+041D	Н	Н	Н	Н	Н	=capital EN, Cyrillic
U+043E	о	о	о	о	о	=small o, Cyrillic
U+041E	О	О	О	О	О	=capital O, Cyrillic
U+043F	п	п	п	п	п	=small pe, Cyrillic
U+041F	П	П	П	П	П	=capital PE, Cyrillic
U+0440	р	р	р	р	р	=small er, Cyrillic
U+0420	Р	Р	Р	Р	Р	=capital ER, Cyrillic
U+0441	с	с	с	с	с	=small es, Cyrillic
U+0421	С	С	С	С	С	=capital ES, Cyrillic
U+0442	т	т	т	т	т	=small te, Cyrillic
U+0422	Т	Т	Т	Т	Т	=capital TE, Cyrillic
U+0443	у	у	у	у	у	=small u, Cyrillic
U+0423	У	У	У	У	У	=capital U, Cyrillic
U+0444	ф	ф	ф	ф	ф	=small ef, Cyrillic
U+0424	Ф	Ф	Ф	Ф	Ф	=capital EF, Cyrillic
U+0445	х	х	х	х	х	=small ha, Cyrillic
U+0425	Х	Х	Х	Х	Х	=capital HA, Cyrillic
U+0446	ц	ц	ц	ц	ц	=small tse, Cyrillic
U+0426	Ц	Ц	Ц	Ц	Ц	=capital TSE, Cyrillic
U+0447	ч	ч	ч	ч	ч	=small che, Cyrillic
U+0427	Ч	Ч	Ч	Ч	Ч	=capital CHE, Cyrillic
U+0448	ш	ш	ш	ш	ш	=small sha, Cyrillic
U+0428	Ш	Ш	Ш	Ш	Ш	=capital SHA, Cyrillic
U+0449	щ	щ	щ	щ	щ	=small shcha, Cyrillic
U+0429	Щ	Щ	Щ	Щ	Щ	=capital SHCHA, Cyrillic
U+044A	ъ	ъ	ъ	ъ	ъ	=small hard sign, Cyrillic
U+042A	Ъ	Ъ	Ъ	Ъ	Ъ	=capital HARD sign, Cyrillic
U+044B	ы	ы	ы	ы	ы	=small yeru, Cyrillic
U+042B	Ы	Ы	Ы	Ы	Ы	=capital YERU, Cyrillic
U+044C	ь	ь	ь	ь	ь	=small soft sign, Cyrillic
U+042C	Ь	Ь	Ь	Ь	Ь	=capital SOFT sign, Cyrillic
U+044D	э	э	э	э	э	=small e, Cyrillic
U+042D	Э	Э	Э	Э	Э	=capital E, Cyrillic
U+044E	ю	ю	ю	ю	ю	=small yu, Cyrillic
U+042E	Ю	Ю	Ю	Ю	Ю	=capital YU, Cyrillic
U+044F	я	я	я	я	я	=small ya, Cyrillic
U+042F	Я	Я	Я	Я	Я	=capital YA, Cyrillic
U+2116	№	№	№	№	№	=numero sign

Tabella u79.17. Entità ISO 8879:1986 ISOcyr2: *non-russian cyrillic*.

Punto di codifica	Macro SGML					Descrizione
U+0452	ђ	ђ	ђ	ђ	ђ	=small dje, Serbian
U+0402	Ђ	Ђ	Ђ	Ђ	Ђ	=capital DJE, Serbian
U+0453	ѓ	ѓ	ѓ	ѓ	ѓ	=small gje, Macedonian
U+0403	Ѓ	Ѓ	Ѓ	Ѓ	Ѓ	=capital GJE Macedonian
U+0454	є	є	є	є	є	=small je, Ukrainian
U+0404	Є	Є	Є	Є	Є	=capital JE, Ukrainian
U+0455	ѕ	ѕ	ѕ	ѕ	ѕ	=small dse, Macedonian
U+0405	Ѕ	S	S	S	S	=capital DSE, Macedonian
U+0456	і	і	і	і	і	=small i, Ukrainian
U+0406	І	І	І	І	І	=capital I, Ukrainian
U+0457	ї	ї	ї	ї	ї	=small yi, Ukrainian
U+0407	Ї	□	□	□	□	=capital YI, Ukrainian
U+0458	ј	ј	ј	ј	ј	=small je, Serbian
U+0408	Ј	J	J	J	J	=capital JE, Serbian
U+0459	љ	љ	љ	љ	љ	=small lje, Serbian
U+0409	Љ	Љ	Љ	Љ	Љ	=capital LJE, Serbian
U+045A	њ	њ	њ	њ	њ	=small nje, Serbian
U+040A	Њ	Њ	Њ	Њ	Њ	=capital NJE, Serbian
U+045B	ћ	ћ	ћ	ћ	ћ	=small tshe, Serbian
U+040B	Ћ	Ћ	Ћ	Ћ	Ћ	=capital TSHE, Serbian
U+045C	ќ	ќ	ќ	ќ	ќ	=small kje Macedonian
U+040C	Ќ	Ќ	Ќ	Ќ	Ќ	=capital KJE, Macedonian
U+045E	ў	ў	ў	ў	ў	=small u, Byelorussian
U+040E	Ў	Ў	Ў	Ў	Ў	=capital U, Byelorussian
U+045F	џ	џ	џ	џ	џ	=small dze, Serbian
U+040F	Џ	Џ	Џ	Џ	Џ	=capital dze, Serbian

Si seleziona il greco con la sigla '**el**'.

Tabella u79.18. Entità ISO 8879:1986 ISOgrk1: *greek letters*.

Punto di codifica	Macro SGML					Descrizione
U+03B1	&agr;	α	α	α	α	=small alpha, Greek
U+0391	&Agr;	Α	Α	Α	Α	greek capital letter alpha
U+03B2	&bgr;	β	β	β	β	=small beta, Greek
U+0392	&Bgr;	Β	Β	Β	Β	greek capital letter beta
U+03B3	&ggr;	γ	γ	γ	γ	=small gamma, Greek
U+0393	&Ggr;	Γ	Γ	Γ	Γ	=capital Gamma, Greek
U+03B4	&dgr;	δ	δ	δ	δ	=small delta, Greek
U+0394	&Dgr;	Δ	Δ	Δ	Δ	=capital Delta, Greek
U+03B5	&egr;	ε	ε	ε	ε	=small epsilon, Greek
U+0395	&Egr;	Ε	Ε	Ε	Ε	greek capital letter epsilon
U+03B6	&zgr;	ζ	ζ	ζ	ζ	=small zeta, Greek
U+0396	&Zgr;	Ζ	Ζ	Ζ	Ζ	greek capital letter zeta
U+03B7	&eegr;	η	η	η	η	=small eta, Greek
U+0397	&EEgr;	Η	Η	Η	Η	greek capital letter eta
U+03B8	&thgr;	θ	θ	θ	θ	=small theta, Greek
U+0398	&THgr;	Θ	Θ	Θ	Θ	=capital Theta, Greek
U+03B9	&igr;	ι	ι	ι	ι	=small iota, Greek
U+0399	&Igr;	Ι	Ι	Ι	Ι	greek capital letter iota
U+03BA	&kgr;	κ	κ	κ	κ	=small kappa, Greek
U+039A	&Kgr;	Κ	Κ	Κ	Κ	greek capital letter kappa
U+03BB	&lgr;	λ	λ	λ	λ	=small lambda, Greek
U+039B	&Lgr;	Λ	Λ	Λ	Λ	=capital Lambda, Greek
U+03BC	&mgr;	μ	μ	μ	μ	=small mu, Greek
U+039C	&Mgr;	Μ	Μ	Μ	Μ	greek capital letter mu
U+03BD	&ngr;	ν	ν	ν	ν	=small nu, Greek
U+039D	&Ngr;	Ν	Ν	Ν	Ν	greek capital letter nu
U+03BE	&xgr;	ξ	ξ	ξ	ξ	=small xi, Greek
U+039E	&Xgr;	Ξ	Ξ	Ξ	Ξ	=capital Xi, Greek
U+03BF	&ogr;	ο	ο	ο	ο	greek small letter omicron
U+039F	&Ogr;	Ο	Ο	Ο	Ο	greek capital letter omicron
U+03C0	&pgr;	π	π	π	π	=small pi, Greek
U+03A0	&Pgr;	Π	Π	Π	Π	=capital Pi, Greek
U+03C1	&rgr;	ρ	ρ	ρ	ρ	=small rho, Greek
U+03A1	&Rgr;	Ρ	Ρ	Ρ	Ρ	greek capital letter rho
U+03C3	&sgr;	σ	σ	σ	σ	=small sigma, Greek
U+03A3	&Sgr;	Σ	Σ	Σ	Σ	=capital Sigma, Greek
U+03C2	&sfgr;	ς	ς	ς	ς	=final small sigma, Greek
U+03C4	&tgr;	τ	τ	τ	τ	=small tau, Greek
U+03A4	&Tgr;	Τ	Τ	Τ	Τ	greek capital letter tau
U+03C5	&ugr;	υ	υ	υ	υ	=small upsilon, Greek
U+03A5	&Ugr;	Υ	Υ	Υ	Υ	greek capital letter upsilon
U+03C6	&phgr;	φ	φ	φ	φ	=small phi, Greek
U+03A6	&PHgr;	Φ	Φ	Φ	Φ	=capital Phi, Greek
U+03C7	&khgr;	χ	χ	χ	χ	=small chi, Greek
U+03A7	&KHgr;	Χ	Χ	Χ	Χ	greek capital letter chi
U+03C8	&psgr;	ψ	ψ	ψ	ψ	=small psi, Greek
U+03A8	&PSgr;	Ψ	Ψ	Ψ	Ψ	=capital Psi, Greek
U+03C9	&ohgr;	ω	ω	ω	ω	=small omega, Greek
U+03A9	&OHgr;	Ω	Ω	Ω	Ω	=capital Omega, Greek

Tabella u79.19. Entità ISO 8879:1986 ISOgrk2: *monotoniko greek*.

Punto di codifica	Macro SGML					Descrizione
U+03AC	&aacgr;	ά	ά	ά	ά	=small alpha, accent, Greek
U+0386	&Aacgr;	Ά	Ά	Ά	Ά	=capital Alpha, accent, Greek
U+03AD	&eacgr;	έ	έ	έ	έ	=small epsilon, accent, Greek
U+0388	&Eacgr;	Έ	Έ	Έ	Έ	=capital Epsilon, accent, Greek
U+03AE	&eeacgr;	ή	ή	ή	ή	=small eta, accent, Greek
U+0389	&EEacgr;	Ή	Ή	Ή	Ή	=capital Eta, accent, Greek
U+03CA	&idigr;	ϊ	ϊ	ϊ	ϊ	=small iota, dieresis, Greek
U+03AA	&Idigr;	Ϊ	Ϊ	Ϊ	Ϊ	=capital Iota, dieresis, Greek
U+03AF	&iacgr;	ί	ί	ί	ί	=small iota, accent, Greek
U+038A	&Iacgr;	Ί	Ί	Ί	Ί	=capital Iota, accent, Greek
U+0390	&idiagr;	ΐ	ΐ	ΐ	ΐ	=small iota, dieresis, accent, Greek
U+03CC	&oacgr;	ό	ό	ό	ό	=small omicron, accent, Greek
U+038C	&Oacgr;	Ό	Ό	Ό	Ό	=capital Omicron, accent, Greek
U+03CB	&udigr;	ϋ	ϋ	ϋ	ϋ	=small upsilon, dieresis, Greek
U+03AB	&Udigr;	Ϋ	Ϋ	Ϋ	Ϋ	=capital Upsilon, dieresis, Greek
U+03CD	&uacgr;	ύ	ύ	ύ	ύ	=small upsilon, accent, Greek
U+038E	&Uacgr;	Ύ	Ύ	Ύ	Ύ	=capital Upsilon, accent, Greek
U+03B0	&udiagr;	ΰ	ΰ	ΰ	ΰ	=small upsilon, dieresis, accent, Greek
U+03CE	&ohacgr;	ώ	ώ	ώ	ώ	=small omega, accent, Greek
U+038F	&OHacgr;	Ώ	Ώ	Ώ	Ώ	=capital Omega, accent, Greek

«

HTML utilizza una propria classificazione delle entità, secondo gli elenchi di questa sezione, includendo anche entità estranee allo standard ISO 8879:1986. L'utilizzo di queste entità è valido nei linguaggi latini; tuttavia, se si scrivono lettere greche utilizzando direttamente il loro codice, si ottiene la loro traslitterazione, a meno di selezionare la lingua greca.

Tabella u79.20. Entità HTML HTMLlat1.

Punto di codifica	Standard	Macro SGML					Descrizione
U+00A0	ISOnum	nbsp;					no-break space = non-breaking space
U+00A1	ISOnum	¡	¡	¡	¡	¡	inverted exclamation mark
U+00A2	ISOnum	¢	¢	¢	¢	¢	cent sign
U+00A3	ISOnum	£	£	£	£	£	pound sign
U+00A4	ISOnum	¤	□	□	□	□	currency sign
U+00A5	ISOnum	¥	¥	¥	¥	¥	yen sign = yuan sign
U+00A6	ISOnum	¦	¦	¦	¦	¦	broken bar = broken vertical bar
U+00A7	ISOnum	§	§	§	§	§	section sign
U+00A8	ISOdia	¨	¨	¨	¨	¨	diaeresis = spacing diaeresis
U+00A9	ISOnum	©	©	©	©	©	copyright sign
U+00AA	ISOnum	ª	ª	ª	ª	ª	feminine ordinal indicator
U+00AB	ISOnum	«	«	«	«	«	left-pointing double angle quotation mark = left pointing guillemet
U+00AC	ISOnum	¬	¬	¬	¬	¬	not sign
U+00AD	ISOnum						soft hyphen = discretionary hyphen
U+00AE	ISOnum	®	®	®	®	®	registered sign = registered trade mark sign
U+00AF	ISOdia	¯	¯	¯	¯	¯	macron = spacing macron = overline = APL overbar
U+00B0	ISOnum	°	°	°	°	°	degree sign
U+00B1	ISOnum	±	±	±	±	±	plus-minus sign = plus-or-minus sign
U+00B2	ISOnum	²	²	²	²	²	superscript two = superscript digit two = squared
U+00B3	ISOnum	³	³	³	³	³	superscript three = superscript digit three = cubed
U+00B4	ISOdia	´	´	´	´	´	acute accent = spacing acute
U+00B5	ISOnum	µ	µ	µ	µ	µ	micro sign
U+00B6	ISOnum	¶	¶	¶	¶	¶	pilcrow sign = paragraph sign
U+00B7	ISOnum	·	·	·	·	·	middle dot = Georgian comma = Greek middle dot
U+00B8	ISOdia	¸	¸	¸	¸	¸	cedilla = spacing cedilla
U+00B9	ISOnum	¹	¹	¹	¹	¹	superscript one = superscript digit one
U+00BA	ISOnum	º	º	º	º	º	masculine ordinal indicator
U+00BB	ISOnum	»	»	»	»	»	right-pointing double angle quotation mark = right pointing guillemet
U+00BC	ISOnum	¼	¼	¼	¼	¼	vulgar fraction one quarter = fraction one quarter
U+00BD	ISOnum	½	½	½	½	½	vulgar fraction one half = fraction one half
U+00BE	ISOnum	¾	¾	¾	¾	¾	vulgar fraction three quarters = fraction three quarters
U+00BF	ISOnum	¿	¿	¿	¿	¿	inverted question mark = turned question mark
U+00C0	ISOlat1	À	À	À	**À**	**À**	latin capital letter A with grave = latin capital letter A grave
U+00C1	ISOlat1	Á	Á	Á	**Á**	**Á**	latin capital letter A with acute
U+00C2	ISOlat1	Â	Â	Â	**Â**	**Â**	latin capital letter A with circumflex
U+00C3	ISOlat1	Ã	Ã	Ã	**Ã**	**Ã**	latin capital letter A with tilde
U+00C4	ISOlat1	Ä	Ä	Ä	**Ä**	**Ä**	latin capital letter A with diaeresis
U+00C5	ISOlat1	Å	Å	Å	**Å**	**Å**	latin capital letter A with ring above = latin capital letter A ring
U+00C6	ISOlat1	Æ	Æ	Æ	**Æ**	**Æ**	latin capital letter AE = latin capital ligature AE
U+00C7	ISOlat1	Ç	Ç	Ç	**Ç**	**Ç**	latin capital letter C with cedilla
U+00C8	ISOlat1	È	È	È	**È**	**È**	latin capital letter E with grave

Punto di codifica	Standard	Macro SGML					Descrizione
U+00C9	ISOlat1	É	É	É	**É**	**É**	latin capital letter E with acute
U+00CA	ISOlat1	Ê	Ê	Ê	**Ê**	**Ê**	latin capital letter E with circumflex
U+00CB	ISOlat1	Ë	Ë	Ë	**Ë**	**Ë**	latin capital letter E with diaeresis
U+00CC	ISOlat1	Ì	Ì	Ì	**Ì**	**Ì**	latin capital letter I with grave
U+00CD	ISOlat1	Í	Í	Í	**Í**	**Í**	latin capital letter I with acute
U+00CE	ISOlat1	Î	Î	Î	**Î**	**Î**	latin capital letter I with circumflex
U+00CF	ISOlat1	Ï	Ï	Ï	**Ï**	**Ï**	latin capital letter I with diaeresis
U+00D0	ISOlat1	Ð	Ð	Ð	**Ð**	**Ð**	latin capital letter ETH
U+00D1	ISOlat1	Ñ	Ñ	Ñ	**Ñ**	**Ñ**	latin capital letter N with tilde
U+00D2	ISOlat1	Ò	Ò	Ò	**Ò**	**Ò**	latin capital letter O with grave
U+00D3	ISOlat1	Ó	Ó	Ó	**Ó**	**Ó**	latin capital letter O with acute
U+00D4	ISOlat1	Ô	Ô	Ô	**Ô**	**Ô**	latin capital letter O with circumflex
U+00D5	ISOlat1	Õ	Õ	Õ	**Õ**	**Õ**	latin capital letter O with tilde
U+00D6	ISOlat1	Ö	Ö	Ö	**Ö**	**Ö**	latin capital letter O with diaeresis
U+00D7	ISOnum	×	×	×	×	×	multiplication sign
U+00D8	ISOlat1	Ø	Ø	Ø	**Ø**	**Ø**	latin capital letter O with stroke = latin capital letter O slash
U+00D9	ISOlat1	Ù	Ù	Ù	**Ù**	**Ù**	latin capital letter U with grave
U+00DA	ISOlat1	Ú	Ú	Ú	**Ú**	**Ú**	latin capital letter U with acute
U+00DB	ISOlat1	Û	Û	Û	**Û**	**Û**	latin capital letter U with circumflex
U+00DC	ISOlat1	Ü	Ü	Ü	**Ü**	**Ü**	latin capital letter U with diaeresis
U+00DD	ISOlat1	Ý	Ý	Ý	**Ý**	**Ý**	latin capital letter Y with acute
U+00DE	ISOlat1	Þ	Þ	Þ	**Þ**	**Þ**	latin capital letter THORN
U+00DF	ISOlat1	ß	ß	ß	**ß**	**ß**	latin small letter sharp s = ess-zed
U+00E0	ISOlat1	à	à	à	**à**	**à**	latin small letter a with grave = latin small letter a grave
U+00E1	ISOlat1	á	á	á	**á**	**á**	latin small letter a with acute
U+00E2	ISOlat1	â	â	â	**â**	**â**	latin small letter a with circumflex
U+00E3	ISOlat1	ã	ã	ã	**ã**	**ã**	latin small letter a with tilde
U+00E4	ISOlat1	ä	ä	ä	**ä**	**ä**	latin small letter a with diaeresis
U+00E5	ISOlat1	å	å	å	**å**	**å**	latin small letter a with ring above = latin small letter a ring
U+00E6	ISOlat1	æ	æ	æ	**æ**	**æ**	latin small letter ae = latin small ligature ae
U+00E7	ISOlat1	ç	ç	ç	**ç**	**ç**	latin small letter c with cedilla
U+00E8	ISOlat1	è	è	è	**è**	**è**	latin small letter e with grave
U+00E9	ISOlat1	é	é	é	**é**	**é**	latin small letter e with acute
U+00EA	ISOlat1	ê	ê	ê	**ê**	**ê**	latin small letter e with circumflex
U+00EB	ISOlat1	ë	ë	ë	**ë**	**ë**	latin small letter e with diaeresis
U+00EC	ISOlat1	ì	ì	ì	**ì**	**ì**	latin small letter i with grave
U+00ED	ISOlat1	í	í	í	**í**	**í**	latin small letter i with acute
U+00EE	ISOlat1	î	î	î	**î**	**î**	latin small letter i with circumflex
U+00EF	ISOlat1	ï	ï	ï	**ï**	**ï**	latin small letter i with diaeresis
U+00F0	ISOlat1	ð	ð	ð	**ð**	**ð**	latin small letter eth
U+00F1	ISOlat1	ñ	ñ	ñ	**ñ**	**ñ**	latin small letter n with tilde
U+00F2	ISOlat1	ò	ò	ò	**ò**	**ò**	latin small letter o with grave
U+00F3	ISOlat1	ó	ó	ó	**ó**	**ó**	latin small letter o with acute
U+00F4	ISOlat1	ô	ô	ô	**ô**	**ô**	latin small letter o with circumflex
U+00F5	ISOlat1	õ	õ	õ	**õ**	**õ**	latin small letter o with tilde
U+00F6	ISOlat1	ö	ö	ö	**ö**	**ö**	latin small letter o with diaeresis
U+00F7	ISOnum	÷	÷	÷	÷	÷	division sign
U+00F8	ISOlat1	ø	ø	ø	**ø**	**ø**	latin small letter o with stroke = latin small letter o slash
U+00F9	ISOlat1	ù	ù	ù	**ù**	**ù**	latin small letter u with grave
U+00FA	ISOlat1	ú	ú	ú	**ú**	**ú**	latin small letter u with acute

Punto di codifica	Standard	Macro SGML					Descrizione
U+00FB	ISOlat1	û	û	*û*	**û**	**û**	latin small letter u with circumflex
U+00FC	ISOlat1	ü	ü	*ü*	**ü**	**ü**	latin small letter u with diaeresis
U+00FD	ISOlat1	ý	ý	*ý*	**ý**	**ý**	latin small letter y with acute
U+00FE	ISOlat1	þ	þ	*þ*	**þ**	**þ**	latin small letter thorn
U+00FF	ISOlat1	ÿ	ÿ	*ÿ*	**ÿ**	**ÿ**	latin small letter y with diaeresis

Tabella u79.21. Entità HTML HTMLspecial.

Punto di codifica	Standard	Macro SGML					Descrizione
U+0022	ISOnum	"	"	*"*	**"**	**"**	quotation mark = APL quote
U+0026	ISOnum	&	&	*&*	**&**	**&**	ampersand
U+003C	ISOnum	<	<	*<*	**<**	**<**	less-than sign
U+003E	ISOnum	>	>	*>*	**>**	**>**	greater-than sign
U+0152	ISOlat2	Œ	Œ	*Œ*	**Œ**	**Œ**	latin capital ligature OE
U+0153	ISOlat2	œ	œ	*œ*	**œ**	**œ**	latin small ligature oe
U+0160	ISOlat2	Š	Š	*Š*	**Š**	**Š**	latin capital letter S with caron
U+0161	ISOlat2	š	š	*š*	**š**	**š**	latin small letter s with caron
U+0178	ISOlat2	Ÿ	Ÿ	*Ÿ*	**Ÿ**	**Ÿ**	latin capital letter Y with diaeresis
U+02C6	ISOpub	ˆ	^	*^*	**^**	**^**	modifier letter circumflex accent
U+02DC	ISOdia	˜	~	*~*	**~**	**~**	small tilde
U+2002	ISOpub						en space
U+2003	ISOpub						em space
U+2009	ISOpub						thin space
U+200C	RFC 2070	‌					zero width non-joiner
U+200D	RFC 2070	‍					zero width joiner
U+200E	RFC 2070	‎	□	□	□	□	left-to-right mark
U+200F	RFC 2070	‏	□	□	□	□	right-to-left mark
U+2013	ISOpub	–	–	–	–	-	en dash
U+2014	ISOpub	—	—	—	—	--	em dash
U+2018	ISOnum	‘	‘	‘	‘	‘	left single quotation mark
U+2019	ISOnum	’	’	’	’	’	right single quotation mark
U+201A		‚	‚	‚	‚	‚	single low-9 quotation mark
U+201C	ISOnum	“	“	“	“	“	left double quotation mark
U+201D	ISOnum	”	”	”	”	”	right double quotation mark
U+201E		„	„	„	„	„	double low-9 quotation mark
U+2020	ISOpub	†	†	*†*	**†**	†	dagger
U+2021	ISOpub	‡	‡	*‡*	**‡**	‡	double dagger
U+2030	ISOtech	‰	‰	‰	‰	‰	per mille sign
U+2039	ISO proposed	‹	‹	‹	‹	‹	single left-pointing angle quotation mark
U+203A	ISO proposed	›	›	›	›	›	single right-pointing angle quotation mark
U+20AC		€	€	€	€	€	euro sign

Tabella u79.22. Entità HTML HTMLsymbol.

Punto di codifica	Standard	Macro SGML					Descrizione
U+0192	ISOtech	ƒ	*f*	*f*	*f*	*f*	latin small f with hook = function= florin
U+0391		Α	A	A	A	A	greek capital letter alpha
U+0392		Β	B	B	B	B	greek capital letter beta
U+0393	ISOgrk3	Γ	Γ	Γ	Γ	Γ	greek capital letter gamma
U+0394	ISOgrk3	Δ	Δ	Δ	Δ	Δ	greek capital letter delta
U+0395		Ε	E	E	E	E	greek capital letter epsilon
U+0396		Ζ	Z	Z	Z	Z	greek capital letter zeta
U+0397		Η	H	H	H	H	greek capital letter eta
U+0398	ISOgrk3	Θ	Θ	Θ	Θ	Θ	greek capital letter theta
U+0399		Ι	I	I	I	I	greek capital letter iota
U+039A		Κ	K	K	K	K	greek capital letter kappa
U+039B	ISOgrk3	Λ	Λ	Λ	Λ	Λ	greek capital letter lambda
U+039C		Μ	M	M	M	M	greek capital letter mu
U+039D		Ν	N	N	N	N	greek capital letter nu
U+039E	ISOgrk3	Ξ	Ξ	Ξ	Ξ	Ξ	greek capital letter xi
U+039F		Ο	O	O	O	O	greek capital letter omicron
U+03A0	ISOgrk3	Π	Π	Π	Π	Π	greek capital letter pi
U+03A1		Ρ	P	P	P	P	greek capital letter rho

Punto di codifica	Standard	Macro SGML					Descrizione
U+03A3	ISOgrk3	Σ	Σ	Σ	Σ	Σ	greek capital letter sigma
U+03A4		Τ	T	T	T	T	greek capital letter tau
U+03A5	ISOgrk3	Υ	Υ	Υ	Υ	Υ	greek capital letter upsilon
U+03A6	ISOgrk3	Φ	Φ	Φ	Φ	Φ	greek capital letter phi
U+03A7		Χ	X	X	X	X	greek capital letter chi
U+03A8	ISOgrk3	Ψ	Ψ	Ψ	Ψ	Ψ	greek capital letter psi
U+03A9	ISOgrk3	Ω	Ω	Ω	Ω	Ω	greek capital letter omega
U+03B1	ISOgrk3	α	α	α	α	α	greek small letter alpha
U+03B2	ISOgrk3	β	β	β	β	β	greek small letter beta
U+03B3	ISOgrk3	γ	γ	γ	γ	γ	greek small letter gamma
U+03B4	ISOgrk3	δ	δ	δ	δ	δ	greek small letter delta
U+03B5	ISOgrk3	ε	ϵ	ϵ	ϵ	ϵ	greek small letter epsilon
U+03B6	ISOgrk3	ζ	ζ	ζ	ζ	ζ	greek small letter zeta
U+03B7	ISOgrk3	η	η	η	η	η	greek small letter eta
U+03B8		θ	θ	θ	θ	θ	greek small letter theta
U+03B9	ISOgrk3	ι	ι	ι	ι	ι	greek small letter iota
U+03BA	ISOgrk3	κ	κ	κ	κ	κ	greek small letter kappa
U+03BB	ISOgrk3	λ	λ	λ	λ	λ	greek small letter lambda
U+03BC	ISOgrk3	μ	μ	μ	μ	μ	greek small letter mu
U+03BD	ISOgrk3	ν	ν	ν	ν	ν	greek small letter nu
U+03BE	ISOgrk3	ξ	ξ	ξ	ξ	ξ	greek small letter xi
U+03BF		ο	o	o	o	o	greek small letter omicron
U+03C0	ISOgrk3	π	π	π	π	π	greek small letter pi
U+03C1	ISOgrk3	ρ	ρ	ρ	ρ	ρ	greek small letter rho
U+03C2	ISOgrk3	ς	ς	ς	ς	ς	greek small letter final sigma
U+03C3	ISOgrk3	σ	σ	σ	σ	σ	greek small letter sigma
U+03C4	ISOgrk3	τ	τ	τ	τ	τ	greek small letter tau
U+03C5	ISOgrk3	υ	υ	υ	υ	υ	greek small letter upsilon
U+03C6		φ	ϕ	ϕ	ϕ	ϕ	greek small letter phi
U+03C7	ISOgrk3	χ	χ	χ	χ	χ	greek small letter chi
U+03C8	ISOgrk3	ψ	ψ	ψ	ψ	ψ	greek small letter psi
U+03C9	ISOgrk3	ω	ω	ω	ω	ω	greek small letter omega
U+03D1		ϑ	ϑ	ϑ	ϑ	ϑ	greek small letter theta symbol
U+03D2		ϒ	ϒ	ϒ	ϒ	ϒ	greek upsilon with hook symbol
U+03D6	ISOgrk3	ϖ	ϖ	ϖ	ϖ	ϖ	greek pi symbol
U+2022	ISOpub	•	•	•	•	•	bullet = black small circle
U+2026	ISOpub	…	…	…	…	…	horizontal ellipsis = three dot leader
U+2032	ISOtech	′	′	′	′	′	prime = minutes = feet
U+2033	ISOtech	″	″	″	″	″	double prime = seconds = inches
U+203E		‾	‾	‾	‾	‾	overline = spacing overscore
U+2044		⁄	/	/	/	/	fraction slash
U+2118	ISOamso	℘	℘	℘	℘	℘	script capital P = power set = Weierstrass p
U+2111	ISOamso	ℑ	ℑ	ℑ	ℑ	ℑ	blackletter capital I = imaginary part
U+211C	ISOamso	ℜ	ℜ	ℜ	ℜ	ℜ	blackletter capital R = real part symbol
U+2122	ISOnum	™	™	™	™	™	trade mark sign
U+2135		ℵ	ℵ	ℵ	ℵ	ℵ	alef symbol = first transfinite cardinal
U+2190	ISOnum	←	←	←	←	←	leftwards arrow
U+2191	ISOnu	↑	↑	↑	↑	↑	upwards arrow
U+2192	ISOnum	→	→	→	→	→	rightwards arrow
U+2193	ISOnum	↓	v	*v*	**v**	**v**	downwards arrow
U+2194	ISOamsa	↔	↔	↔	↔	↔	left right arrow
U+21B5		↵	↵	↵	↵	↵	downwards arrow with corner leftwards = carriage return
U+21D0	ISOtech	⇐	⇐	⇐	⇐	⇐	leftwards double arrow
U+21D1	ISOamsa	⇑	⇑	⇑	⇑	⇑	upwards double arrow
U+21D2	ISOtech	⇒	⇒	⇒	⇒	⇒	rightwards double arrow
U+21D3	ISOamsa	⇓	⇓	⇓	⇓	⇓	downwards double arrow
U+21D4	ISOamsa	⇔	⇔	⇔	⇔	⇔	left right double arrow
U+2200	ISOtech	∀	∀	∀	∀	∀	for all
U+2202	ISOtech	∂	∂	∂	∂	∂	partial differential
U+2203	ISOtech	∃	∃	∃	∃	∃	there exists
U+2205	ISOamso	∅	∅	∅	∅	∅	empty set = null set = diameter
U+2207	ISOtech	∇	∇	∇	∇	∇	nabla = backward difference
U+220A	ISOtech	∈	□	□	□	□	element of

Punto di codifica	Standard	Macro SGML					Descrizione
U+2209	ISOtech	∉	∉	∉	∉	∉	not an element of
U+220D	ISOtech	∋	∋	∋	∋	∋	contains as member
U+220F	ISOamsb	∏	∏	∏	∏	∏	n-ary product = product sign
U+2211	ISOamsb	∑	∑	∑	∑	∑	n-ary sumation
U+2212	ISOtech	−	−	−	−	−	minus sign
U+2217	ISOtech	∗	∗	∗	∗	∗	asterisk operator
U+221A	ISOtech	√	√	√	√	√	square root = radical sign
U+221D	ISOtech	∝	∝	∝	∝	∝	proportional to
U+221E	ISOtech	∞	∞	∞	∞	∞	infinity
U+2220	ISOamso	∠	∠	∠	∠	∠	angle
U+2227	ISOtech	∧	∧	∧	∧	∧	logical and = wedge
U+2228	ISOtech	∨	∨	∨	∨	∨	logical or = vee
U+2229	ISOtech	∩	∩	∩	∩	∩	intersection = cap
U+222A	ISOtech	∪	∪	∪	∪	∪	union = cup
U+222B	ISOtech	∫	∫	∫	∫	∫	integral
U+2234	ISOtech	∴	∴	∴	∴	∴	therefore
U+223C	ISOtech	∼	∼	∼	∼	∼	tilde operator = varies with = similar to
U+2245	ISOtech	≅	≅	≅	≅	≅	approximately equal to
U+2248	ISOamsr	≈	≍	≍	≍	≍	almost equal to = asymptotic to
U+2260	ISOtech	≠	≠	≠	≠	≠	not equal to
U+2261	ISOtech	≡	≡	≡	≡	≡	identical to
U+2264	ISOtech	≤	≤	≤	≤	≤	less-than or equal to
U+2265	ISOtech	≥	≥	≥	≥	≥	greater-than or equal to
U+2282	ISOtech	⊂	⊂	⊂	⊂	⊂	subset of
U+2283	ISOtech	⊃	⊃	⊃	⊃	⊃	superset of
U+2284	ISOamsn	⊄	⊄	⊄	⊄	⊄	not a subset of
U+2286	ISOtech	⊆	⊆	⊆	⊆	⊆	subset of or equal to
U+2287	ISOtech	⊇	⊇	⊇	⊇	⊇	superset of or equal to
U+2295	ISOamsb	⊕	⊕	⊕	⊕	⊕	circled plus = direct sum
U+2297	ISOamsb	⊗	⊗	⊗	⊗	⊗	circled times = vector product
U+22A5	ISOtech	⊥	⊥	⊥	⊥	⊥	up tack = orthogonal to = perpendicular
U+22C5	ISOamsb	⋅	⋅	⋅	⋅	⋅	dot operator
U+2308	ISOamsc	⌈	⌈	⌈	⌈	⌈	left ceiling = apl upstile
U+2309	ISOamsc	⌉	⌉	⌉	⌉	⌉	right ceiling
U+230A	ISOamsc	⌊	⌊	⌊	⌊	⌊	left floor = apl downstile
U+230B	ISOamsc	⌋	⌋	⌋	⌋	⌋	right floor
U+2329	ISOtech	⟨	□	□	□	□	left-pointing angle bracket = bra
U+232A	ISOtech	⟩	□	□	□	□	right-pointing angle bracket = ket
U+25CA	ISOpub	◊	◊	◊	◊	◊	lozenge
U+2660	ISOpub	♠	♠	♠	♠	♠	black spade suit
U+2663	ISOpub	♣	♣	♣	♣	♣	black club suit = shamrock
U+2665	ISOpub	♥	♥	♥	♥	♥	black heart suit = valentine
U+2666	ISOpub	♦	♦	♦	♦	♦	black diamond suit

Riferimenti

«

- *Unicode home page*
  http://www.unicode.org/
- *Unicode character database*
  http://www.unicode.org/Public/UNIDATA/
- Vidar Bronken Gundersen, Rune Mathisen, *SGML/XML character entity reference*
  http://www.bitjungle.com/isoent/

Riferimenti 535

Per maggiore comodità, viene riportato un elenco dei simboli gestiti da Alml, ordinato secondo i punti di codifica.

Si ricorda che le attribuzioni ai punti di codifica possono essere errate, pertanto potrebbero cambiare in futuro.

I simboli associati ai punti di codifica non sono sempre perfettamente adeguati agli standard; la forma esatta dei simboli si può verificare presso *http://www.unicode.org/charts/* .

Tabella u80.1. *C0 Controls and basic latin.*

Punto di codifica	Aspetto	Descrizione
U+0000		NULL
U+0001		START OF HEADING
U+0002		START OF TEXT
U+0003		END OF TEXT
U+0004		END OF TRANSMISSION
U+0005		ENQUIRY
U+0006		ACKNOWLEDGE
U+0007		BELL
U+0008		BACKSPACE
U+0009		CHARACTER TABULATION
U+000A		LINE FEED (LF)
U+000B		LINE TABULATION
U+000C		FORM FEED (FF)
U+000D		CARRIAGE RETURN (CR)
U+000E		SHIFT OUT
U+000F		SHIFT IN
U+0010		DATA LINK ESCAPE
U+0011		DEVICE CONTROL ONE
U+0012		DEVICE CONTROL TWO
U+0013		DEVICE CONTROL THREE
U+0014		DEVICE CONTROL FOUR
U+0015		NEGATIVE ACKNOWLEDGE
U+0016		SYNCHRONOUS IDLE
U+0017		END OF TRANSMISSION BLOCK
U+0018		CANCEL
U+0019		END OF MEDIUM
U+001A		SUBSTITUTE
U+001B		ESCAPE
U+001C		INFORMATION SEPARATOR FOUR
U+001D		INFORMATION SEPARATOR THREE
U+001E		INFORMATION SEPARATOR TWO
U+001F		INFORMATION SEPARATOR ONE
U+0020		SPACE
U+0021	!	EXCLAMATION MARK
U+0022	"	QUOTATION MARK
U+0023	#	NUMBER SIGN
U+0024	$	DOLLAR SIGN
U+0025	%	PERCENT SIGN
U+0026	&	AMPERSAND
U+0027	’	APOSTROPHE
U+0028	(	LEFT PARENTHESIS
U+0029	)	RIGHT PARENTHESIS
U+002A	*	ASTERISK
U+002B	+	PLUS SIGN
U+002C	,	COMMA
U+002D	-	HYPHEN-MINUS
U+002E	.	FULL STOP
U+002F	/	SOLIDUS
U+0030	0	DIGIT ZERO
U+0031	1	DIGIT ONE
U+0032	2	DIGIT TWO
U+0033	3	DIGIT THREE
U+0034	4	DIGIT FOUR
U+0035	5	DIGIT FIVE
U+0036	6	DIGIT SIX
U+0037	7	DIGIT SEVEN
U+0038	8	DIGIT EIGHT
U+0039	9	DIGIT NINE
U+003A	:	COLON
U+003B	;	SEMICOLON
U+003C	<	LESS-THAN SIGN
U+003D	=	EQUALS SIGN
U+003E	>	GREATER-THAN SIGN
U+003F	?	QUESTION MARK
U+0040	@	COMMERCIAL AT
U+0041	A	LATIN CAPITAL LETTER A

Punto di codifica	Aspetto	Descrizione
U+0042	B	LATIN CAPITAL LETTER B
U+0043	C	LATIN CAPITAL LETTER C
U+0044	D	LATIN CAPITAL LETTER D
U+0045	E	LATIN CAPITAL LETTER E
U+0046	F	LATIN CAPITAL LETTER F
U+0047	G	LATIN CAPITAL LETTER G
U+0048	H	LATIN CAPITAL LETTER H
U+0049	I	LATIN CAPITAL LETTER I
U+004A	J	LATIN CAPITAL LETTER J
U+004B	K	LATIN CAPITAL LETTER K
U+004C	L	LATIN CAPITAL LETTER L
U+004D	M	LATIN CAPITAL LETTER M
U+004E	N	LATIN CAPITAL LETTER N
U+004F	O	LATIN CAPITAL LETTER O
U+0050	P	LATIN CAPITAL LETTER P
U+0051	Q	LATIN CAPITAL LETTER Q
U+0052	R	LATIN CAPITAL LETTER R
U+0053	S	LATIN CAPITAL LETTER S
U+0054	T	LATIN CAPITAL LETTER T
U+0055	U	LATIN CAPITAL LETTER U
U+0056	V	LATIN CAPITAL LETTER V
U+0057	W	LATIN CAPITAL LETTER W
U+0058	X	LATIN CAPITAL LETTER X
U+0059	Y	LATIN CAPITAL LETTER Y
U+005A	Z	LATIN CAPITAL LETTER Z
U+005B	[	LEFT SQUARE BRACKET
U+005C	\	REVERSE SOLIDUS
U+005D	]	RIGHT SQUARE BRACKET
U+005E	^	CIRCUMFLEX ACCENT
U+005F	–	LOW LINE
U+0060	‘	GRAVE ACCENT
U+0061	a	LATIN SMALL LETTER A
U+0062	b	LATIN SMALL LETTER B
U+0063	c	LATIN SMALL LETTER C
U+0064	d	LATIN SMALL LETTER D
U+0065	e	LATIN SMALL LETTER E
U+0066	f	LATIN SMALL LETTER F
U+0067	g	LATIN SMALL LETTER G
U+0068	h	LATIN SMALL LETTER H
U+0069	i	LATIN SMALL LETTER I
U+006A	j	LATIN SMALL LETTER J
U+006B	k	LATIN SMALL LETTER K
U+006C	l	LATIN SMALL LETTER L
U+006D	m	LATIN SMALL LETTER M
U+006E	n	LATIN SMALL LETTER N
U+006F	o	LATIN SMALL LETTER O
U+0070	p	LATIN SMALL LETTER P
U+0071	q	LATIN SMALL LETTER Q
U+0072	r	LATIN SMALL LETTER R
U+0073	s	LATIN SMALL LETTER S
U+0074	t	LATIN SMALL LETTER T
U+0075	u	LATIN SMALL LETTER U
U+0076	v	LATIN SMALL LETTER V
U+0077	w	LATIN SMALL LETTER W
U+0078	x	LATIN SMALL LETTER X
U+0079	y	LATIN SMALL LETTER Y
U+007A	z	LATIN SMALL LETTER Z
U+007B	{	LEFT CURLY BRACKET
U+007C	|	VERTICAL LINE
U+007D	}	RIGHT CURLY BRACKET
U+007E	~	TILDE
U+007F		DELETE

Tabella u80.2. *C1 Controls and latin-1 supplement.*

Punto di codifica	Aspetto	Descrizione
U+0080		
U+0081		
U+0082		BREAK PERMITTED HERE
U+0083		NO BREAK HERE
U+0084		
U+0085		NEXT LINE (NEL)
U+0086		START OF SELECTED AREA
U+0087		END OF SELECTED AREA
U+0088		CHARACTER TABULATION SET
U+0089		CHARACTER TABULATION WITH JUSTIFICATION
U+008A		LINE TABULATION SET
U+008B		PARTIAL LINE FORWARD
U+008C		PARTIAL LINE BACKWARD
U+008D		REVERSE LINE FEED
U+008E		SINGLE SHIFT TWO
U+008F		SINGLE SHIFT THREE
U+0090		DEVICE CONTROL STRING
U+0091		PRIVATE USE ONE
U+0092		PRIVATE USE TWO

Punto di codifica	Aspetto	Descrizione
U+0093		SET TRANSMIT STATE
U+0094		CANCEL CHARACTER
U+0095		MESSAGE WAITING
U+0096		START OF GUARDED AREA
U+0097		END OF GUARDED AREA
U+0098		START OF STRING
U+0099		
U+009A		SINGLE CHARACTER INTRODUCER
U+009B		CONTROL SEQUENCE INTRODUCER
U+009C		STRING TERMINATOR
U+009D		OPERATING SYSTEM COMMAND
U+009E		PRIVACY MESSAGE
U+009F		APPLICATION PROGRAM COMMAND
U+00A0		NO-BREAK SPACE
U+00A1	¡	INVERTED EXCLAMATION MARK
U+00A2	¢	CENT SIGN
U+00A3	£	POUND SIGN
U+00A4	□	CURRENCY SIGN
U+00A5	¥	YEN SIGN
U+00A6	¦	BROKEN BAR
U+00A7	§	SECTION SIGN
U+00A8	¨	DIAERESIS
U+00A9	©	COPYRIGHT SIGN
U+00AA	ª	FEMININE ORDINAL INDICATOR
U+00AB	«	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
U+00AC	¬	NOT SIGN
U+00AD		SOFT HYPHEN
U+00AE	®	REGISTERED SIGN
U+00AF	¯	MACRON
U+00B0	°	DEGREE SIGN
U+00B1	±	PLUS-MINUS SIGN
U+00B2	²	SUPERSCRIPT TWO
U+00B3	³	SUPERSCRIPT THREE
U+00B4	´	ACUTE ACCENT
U+00B5	µ	MICRO SIGN
U+00B6	¶	PILCROW SIGN
U+00B7	·	MIDDLE DOT
U+00B8	¸	CEDILLA
U+00B9	¹	SUPERSCRIPT ONE
U+00BA	º	MASCULINE ORDINAL INDICATOR
U+00BB	»	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
U+00BC	¼	VULGAR FRACTION ONE QUARTER
U+00BD	½	VULGAR FRACTION ONE HALF
U+00BE	¾	VULGAR FRACTION THREE QUARTERS
U+00BF	¿	INVERTED QUESTION MARK
U+00C0	À	LATIN CAPITAL LETTER A WITH GRAVE
U+00C1	Á	LATIN CAPITAL LETTER A WITH ACUTE
U+00C2	Â	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
U+00C3	Ã	LATIN CAPITAL LETTER A WITH TILDE
U+00C4	Ä	LATIN CAPITAL LETTER A WITH DIAERESIS
U+00C5	Å	LATIN CAPITAL LETTER A WITH RING ABOVE
U+00C6	Æ	LATIN CAPITAL LETTER AE (ash)
U+00C7	Ç	LATIN CAPITAL LETTER C WITH CEDILLA
U+00C8	È	LATIN CAPITAL LETTER E WITH GRAVE
U+00C9	É	LATIN CAPITAL LETTER E WITH ACUTE
U+00CA	Ê	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
U+00CB	Ë	LATIN CAPITAL LETTER E WITH DIAERESIS
U+00CC	Ì	LATIN CAPITAL LETTER I WITH GRAVE
U+00CD	Í	LATIN CAPITAL LETTER I WITH ACUTE
U+00CE	Î	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
U+00CF	Ï	LATIN CAPITAL LETTER I WITH DIAERESIS
U+00D0	Ð	LATIN CAPITAL LETTER ETH (Icelandic)
U+00D1	Ñ	LATIN CAPITAL LETTER N WITH TILDE
U+00D2	Ò	LATIN CAPITAL LETTER O WITH GRAVE
U+00D3	Ó	LATIN CAPITAL LETTER O WITH ACUTE
U+00D4	Ô	LATIN CAPITAL LETTER O WITH CIRCUMFLEX
U+00D5	Õ	LATIN CAPITAL LETTER O WITH TILDE
U+00D6	Ö	LATIN CAPITAL LETTER O WITH DIAERESIS
U+00D7	×	MULTIPLICATION SIGN
U+00D8	Ø	LATIN CAPITAL LETTER O WITH STROKE
U+00D9	Ù	LATIN CAPITAL LETTER U WITH GRAVE
U+00DA	Ú	LATIN CAPITAL LETTER U WITH ACUTE
U+00DB	Û	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
U+00DC	Ü	LATIN CAPITAL LETTER U WITH DIAERESIS
U+00DD	Ý	LATIN CAPITAL LETTER Y WITH ACUTE
U+00DE	Þ	LATIN CAPITAL LETTER THORN (Icelandic)
U+00DF	ß	LATIN SMALL LETTER SHARP S (German)
U+00E0	à	LATIN SMALL LETTER A WITH GRAVE

Punto di codifica	Aspetto	Descrizione
U+00E1	á	LATIN SMALL LETTER A WITH ACUTE
U+00E2	â	LATIN SMALL LETTER A WITH CIRCUMFLEX
U+00E3	ã	LATIN SMALL LETTER A WITH TILDE
U+00E4	ä	LATIN SMALL LETTER A WITH DIAERESIS
U+00E5	å	LATIN SMALL LETTER A WITH RING ABOVE
U+00E6	æ	LATIN SMALL LETTER AE (ash)
U+00E7	ç	LATIN SMALL LETTER C WITH CEDILLA
U+00E8	è	LATIN SMALL LETTER E WITH GRAVE
U+00E9	é	LATIN SMALL LETTER E WITH ACUTE
U+00EA	ê	LATIN SMALL LETTER E WITH CIRCUMFLEX
U+00EB	ë	LATIN SMALL LETTER E WITH DIAERESIS
U+00EC	ì	LATIN SMALL LETTER I WITH GRAVE
U+00ED	í	LATIN SMALL LETTER I WITH ACUTE
U+00EE	î	LATIN SMALL LETTER I WITH CIRCUMFLEX
U+00EF	ï	LATIN SMALL LETTER I WITH DIAERESIS
U+00F0	ð	LATIN SMALL LETTER ETH (Icelandic)
U+00F1	ñ	LATIN SMALL LETTER N WITH TILDE
U+00F2	ò	LATIN SMALL LETTER O WITH GRAVE
U+00F3	ó	LATIN SMALL LETTER O WITH ACUTE
U+00F4	ô	LATIN SMALL LETTER O WITH CIRCUMFLEX
U+00F5	õ	LATIN SMALL LETTER O WITH TILDE
U+00F6	ö	LATIN SMALL LETTER O WITH DIAERESIS
U+00F7	÷	DIVISION SIGN
U+00F8	ø	LATIN SMALL LETTER O WITH STROKE
U+00F9	ù	LATIN SMALL LETTER U WITH GRAVE
U+00FA	ú	LATIN SMALL LETTER U WITH ACUTE
U+00FB	û	LATIN SMALL LETTER U WITH CIRCUMFLEX
U+00FC	ü	LATIN SMALL LETTER U WITH DIAERESIS
U+00FD	ý	LATIN SMALL LETTER Y WITH ACUTE
U+00FE	þ	LATIN SMALL LETTER THORN (Icelandic)
U+00FF	ÿ	LATIN SMALL LETTER Y WITH DIAERESIS

Tabella u80.3. *Latin extended-A.*

Punto di codifica	Aspetto	Descrizione
U+0100	Ā	LATIN CAPITAL LETTER A WITH MACRON
U+0101	ā	LATIN SMALL LETTER A WITH MACRON
U+0102	Ă	LATIN CAPITAL LETTER A WITH BREVE
U+0103	ă	LATIN SMALL LETTER A WITH BREVE
U+0104	Ą	LATIN CAPITAL LETTER A WITH OGONEK
U+0105	ą	LATIN SMALL LETTER A WITH OGONEK
U+0106	Ć	LATIN CAPITAL LETTER C WITH ACUTE
U+0107	ć	LATIN SMALL LETTER C WITH ACUTE
U+0108	Ĉ	LATIN CAPITAL LETTER C WITH CIRCUMFLEX
U+0109	ĉ	LATIN SMALL LETTER C WITH CIRCUMFLEX
U+010A	Ċ	LATIN CAPITAL LETTER C WITH DOT ABOVE
U+010B	ċ	LATIN SMALL LETTER C WITH DOT ABOVE
U+010C	Č	LATIN CAPITAL LETTER C WITH CARON
U+010D	č	LATIN SMALL LETTER C WITH CARON
U+010E	Ď	LATIN CAPITAL LETTER D WITH CARON
U+010F	ď	LATIN SMALL LETTER D WITH CARON
U+0110	Đ	LATIN CAPITAL LETTER D WITH STROKE
U+0111	đ	LATIN SMALL LETTER D WITH STROKE
U+0112	Ē	LATIN CAPITAL LETTER E WITH MACRON
U+0113	ē	LATIN SMALL LETTER E WITH MACRON
U+0114	Ĕ	LATIN CAPITAL LETTER E WITH BREVE
U+0115	ĕ	LATIN SMALL LETTER E WITH BREVE
U+0116	Ė	LATIN CAPITAL LETTER E WITH DOT ABOVE
U+0117	ė	LATIN SMALL LETTER E WITH DOT ABOVE
U+0118	Ę	LATIN CAPITAL LETTER E WITH OGONEK
U+0119	ę	LATIN SMALL LETTER E WITH OGONEK
U+011A	Ě	LATIN CAPITAL LETTER E WITH CARON
U+011B	ě	LATIN SMALL LETTER E WITH CARON
U+011C	Ĝ	LATIN CAPITAL LETTER G WITH CIRCUMFLEX
U+011D	ĝ	LATIN SMALL LETTER G WITH CIRCUMFLEX
U+011E	Ğ	LATIN CAPITAL LETTER G WITH BREVE
U+011F	ğ	LATIN SMALL LETTER G WITH BREVE
U+0120	Ġ	LATIN CAPITAL LETTER G WITH DOT ABOVE
U+0121	ġ	LATIN SMALL LETTER G WITH DOT ABOVE
U+0122	Ģ	LATIN CAPITAL LETTER G WITH CEDILLA
U+0123	□	LATIN SMALL LETTER G WITH CEDILLA
U+0124	Ĥ	LATIN CAPITAL LETTER H WITH CIRCUMFLEX
U+0125	ĥ	LATIN SMALL LETTER H WITH CIRCUMFLEX
U+0126	Ħ	LATIN CAPITAL LETTER H WITH STROKE
U+0127	ħ	LATIN SMALL LETTER H WITH STROKE
U+0128	Ĩ	LATIN CAPITAL LETTER I WITH TILDE
U+0129	ĩ	LATIN SMALL LETTER I WITH TILDE
U+012A	Ī	LATIN CAPITAL LETTER I WITH MACRON
U+012B	ī	LATIN SMALL LETTER I WITH MACRON
U+012C	Ĭ	LATIN CAPITAL LETTER I WITH BREVE
U+012D	ĭ	LATIN SMALL LETTER I WITH BREVE
U+012E	Į	LATIN CAPITAL LETTER I WITH OGONEK
U+012F	į	LATIN SMALL LETTER I WITH OGONEK

Punto di codifica	Aspetto	Descrizione
U+0130	İ	LATIN CAPITAL LETTER I WITH DOT ABOVE
U+0131	ı	LATIN SMALL LETTER DOTLESS I
U+0132	IJ	LATIN CAPITAL LIGATURE IJ
U+0133	ij	LATIN SMALL LIGATURE IJ
U+0134	Ĵ	LATIN CAPITAL LETTER J WITH CIRCUMFLEX
U+0135	ĵ	LATIN SMALL LETTER J WITH CIRCUMFLEX
U+0136	Ķ	LATIN CAPITAL LETTER K WITH CEDILLA
U+0137	ķ	LATIN SMALL LETTER K WITH CEDILLA
U+0138	□	LATIN SMALL LETTER KRA (Greenlandic)
U+0139	Ĺ	LATIN CAPITAL LETTER L WITH ACUTE
U+013A	ĺ	LATIN SMALL LETTER L WITH ACUTE
U+013B	Ļ	LATIN CAPITAL LETTER L WITH CEDILLA
U+013C	ļ	LATIN SMALL LETTER L WITH CEDILLA
U+013D	L’	LATIN CAPITAL LETTER L WITH CARON
U+013E	l’	LATIN SMALL LETTER L WITH CARON
U+013F	L·	LATIN CAPITAL LETTER L WITH MIDDLE DOT
U+0140	l·	LATIN SMALL LETTER L WITH MIDDLE DOT
U+0141	Ł	LATIN CAPITAL LETTER L WITH STROKE
U+0142	ł	LATIN SMALL LETTER L WITH STROKE
U+0143	Ń	LATIN CAPITAL LETTER N WITH ACUTE
U+0144	ń	LATIN SMALL LETTER N WITH ACUTE
U+0145	Ņ	LATIN CAPITAL LETTER N WITH CEDILLA
U+0146	ņ	LATIN SMALL LETTER N WITH CEDILLA
U+0147	Ň	LATIN CAPITAL LETTER N WITH CARON
U+0148	ň	LATIN SMALL LETTER N WITH CARON
U+0149	□	LATIN SMALL LETTER N PRECEDED BY APOSTROPHE
U+014A	■	LATIN CAPITAL LETTER ENG (Sami)
U+014B	■	LATIN SMALL LETTER ENG (Sami)
U+014C	Ō	LATIN CAPITAL LETTER O WITH MACRON
U+014D	ō	LATIN SMALL LETTER O WITH MACRON
U+014E	Ŏ	LATIN CAPITAL LETTER O WITH BREVE
U+014F	ŏ	LATIN SMALL LETTER O WITH BREVE
U+0150	Ő	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE
U+0151	ő	LATIN SMALL LETTER O WITH DOUBLE ACUTE
U+0152	Œ	LATIN CAPITAL LIGATURE OE
U+0153	œ	LATIN SMALL LIGATURE OE
U+0154	Ŕ	LATIN CAPITAL LETTER R WITH ACUTE
U+0155	ŕ	LATIN SMALL LETTER R WITH ACUTE
U+0156	Ŗ	LATIN CAPITAL LETTER R WITH CEDILLA
U+0157	ŗ	LATIN SMALL LETTER R WITH CEDILLA
U+0158	Ř	LATIN CAPITAL LETTER R WITH CARON
U+0159	ř	LATIN SMALL LETTER R WITH CARON
U+015A	Ś	LATIN CAPITAL LETTER S WITH ACUTE
U+015B	ś	LATIN SMALL LETTER S WITH ACUTE
U+015C	Ŝ	LATIN CAPITAL LETTER S WITH CIRCUMFLEX
U+015D	ŝ	LATIN SMALL LETTER S WITH CIRCUMFLEX
U+015E	Ş	LATIN CAPITAL LETTER S WITH CEDILLA
U+015F	ş	LATIN SMALL LETTER S WITH CEDILLA
U+0160	Š	LATIN CAPITAL LETTER S WITH CARON
U+0161	š	LATIN SMALL LETTER S WITH CARON
U+0162	Ţ	LATIN CAPITAL LETTER T WITH CEDILLA
U+0163	ţ	LATIN SMALL LETTER T WITH CEDILLA
U+0164	Ť	LATIN CAPITAL LETTER T WITH CARON
U+0165	t’	LATIN SMALL LETTER T WITH CARON
U+0166	Ŧ	LATIN CAPITAL LETTER T WITH STROKE
U+0167	ŧ	LATIN SMALL LETTER T WITH STROKE
U+0168	Ũ	LATIN CAPITAL LETTER U WITH TILDE
U+0169	ũ	LATIN SMALL LETTER U WITH TILDE
U+016A	Ū	LATIN CAPITAL LETTER U WITH MACRON
U+016B	ū	LATIN SMALL LETTER U WITH MACRON
U+016C	Ŭ	LATIN CAPITAL LETTER U WITH BREVE
U+016D	ŭ	LATIN SMALL LETTER U WITH BREVE
U+016E	Ů	LATIN CAPITAL LETTER U WITH RING ABOVE
U+016F	ů	LATIN SMALL LETTER U WITH RING ABOVE
U+0170	Ű	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE
U+0171	ű	LATIN SMALL LETTER U WITH DOUBLE ACUTE
U+0172	Ų	LATIN CAPITAL LETTER U WITH OGONEK
U+0173	ų	LATIN SMALL LETTER U WITH OGONEK
U+0174	Ŵ	LATIN CAPITAL LETTER W WITH CIRCUMFLEX
U+0175	ŵ	LATIN SMALL LETTER W WITH CIRCUMFLEX
U+0176	Ŷ	LATIN CAPITAL LETTER Y WITH CIRCUMFLEX
U+0177	ŷ	LATIN SMALL LETTER Y WITH CIRCUMFLEX
U+0178	Ÿ	LATIN CAPITAL LETTER Y WITH DIAERESIS
U+0179	Ź	LATIN CAPITAL LETTER Z WITH ACUTE
U+017A	ź	LATIN SMALL LETTER Z WITH ACUTE
U+017B	Ż	LATIN CAPITAL LETTER Z WITH DOT ABOVE
U+017C	ż	LATIN SMALL LETTER Z WITH DOT ABOVE
U+017D	Ž	LATIN CAPITAL LETTER Z WITH CARON
U+017E	ž	LATIN SMALL LETTER Z WITH CARON
U+017F	s	LATIN SMALL LETTER LONG S

Tabella u80.4. *Greek and Coptic*.

Punto di codifica	Aspetto	Descrizione
U+0374	ʹ	GREEK NUMERAL SIGN (Dexia keraia)
U+0375	͵	GREEK LOWER NUMERAL SIGN (Aristeri keraia)
U+037A	ͺ	GREEK YPOGEGRAMMENI
U+037E	;	GREEK QUESTION MARK (Erotimatiko)
U+0384	΄	GREEK TONOS
U+0385	΅	GREEK DIALYTIKA TONOS
U+0386	Ά	GREEK CAPITAL LETTER ALPHA WITH TONOS
U+0387	·	GREEK ANO TELEIA
U+0388	Έ	GREEK CAPITAL LETTER EPSILON WITH TONOS
U+0389	Ή	GREEK CAPITAL LETTER ETA WITH TONOS
U+038A	Ί	GREEK CAPITAL LETTER IOTA WITH TONOS
U+038C	Ό	GREEK CAPITAL LETTER OMICRON WITH TONOS
U+038E	Ύ	GREEK CAPITAL LETTER UPSILON WITH TONOS
U+038F	Ώ	GREEK CAPITAL LETTER OMEGA WITH TONOS
U+0390	ΐ	GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS
U+0391	Α	GREEK CAPITAL LETTER ALPHA
U+0392	Β	GREEK CAPITAL LETTER BETA
U+0393	Γ	GREEK CAPITAL LETTER GAMMA
U+0394	Δ	GREEK CAPITAL LETTER DELTA
U+0395	Ε	GREEK CAPITAL LETTER EPSILON
U+0396	Ζ	GREEK CAPITAL LETTER ZETA
U+0397	Η	GREEK CAPITAL LETTER ETA
U+0398	Θ	GREEK CAPITAL LETTER THETA
U+0399	Ι	GREEK CAPITAL LETTER IOTA
U+039A	Κ	GREEK CAPITAL LETTER KAPPA
U+039B	Λ	GREEK CAPITAL LETTER LAMDA
U+039C	Μ	GREEK CAPITAL LETTER MU
U+039D	Ν	GREEK CAPITAL LETTER NU
U+039E	Ξ	GREEK CAPITAL LETTER XI
U+039F	Ο	GREEK CAPITAL LETTER OMICRON
U+03A0	Π	GREEK CAPITAL LETTER PI
U+03A1	Ρ	GREEK CAPITAL LETTER RHO
U+03A3	Σ	GREEK CAPITAL LETTER SIGMA
U+03A4	Τ	GREEK CAPITAL LETTER TAU
U+03A5	Υ	GREEK CAPITAL LETTER UPSILON
U+03A6	Φ	GREEK CAPITAL LETTER PHI
U+03A7	Χ	GREEK CAPITAL LETTER CHI
U+03A8	Ψ	GREEK CAPITAL LETTER PSI
U+03A9	Ω	GREEK CAPITAL LETTER OMEGA
U+03AA	Ϊ	GREEK CAPITAL LETTER IOTA WITH DIALYTIKA
U+03AB	Ϋ	GREEK CAPITAL LETTER UPSILON WITH DIALYTIKA
U+03AC	ά	GREEK SMALL LETTER ALPHA WITH TONOS
U+03AD	έ	GREEK SMALL LETTER EPSILON WITH TONOS
U+03AE	ή	GREEK SMALL LETTER ETA WITH TONOS
U+03AF	ί	GREEK SMALL LETTER IOTA WITH TONOS
U+03B0	ΰ	GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND TONOS
U+03B1	α	GREEK SMALL LETTER ALPHA
U+03B2	β	GREEK SMALL LETTER BETA
U+03B3	γ	GREEK SMALL LETTER GAMMA
U+03B4	δ	GREEK SMALL LETTER DELTA
U+03B5	ε	GREEK SMALL LETTER EPSILON
U+03B6	ζ	GREEK SMALL LETTER ZETA
U+03B7	η	GREEK SMALL LETTER ETA
U+03B8	θ	GREEK SMALL LETTER THETA
U+03B9	ι	GREEK SMALL LETTER IOTA
U+03BA	κ	GREEK SMALL LETTER KAPPA
U+03BB	λ	GREEK SMALL LETTER LAMDA
U+03BC	μ	GREEK SMALL LETTER MU
U+03BD	ν	GREEK SMALL LETTER NU
U+03BE	ξ	GREEK SMALL LETTER XI
U+03BF	ο	GREEK SMALL LETTER OMICRON
U+03C0	π	GREEK SMALL LETTER PI
U+03C1	ρ	GREEK SMALL LETTER RHO
U+03C2	ς	GREEK SMALL LETTER FINAL SIGMA
U+03C3	σ	GREEK SMALL LETTER SIGMA
U+03C4	τ	GREEK SMALL LETTER TAU
U+03C5	υ	GREEK SMALL LETTER UPSILON
U+03C6	φ	GREEK SMALL LETTER PHI
U+03C7	χ	GREEK SMALL LETTER CHI
U+03C8	ψ	GREEK SMALL LETTER PSI
U+03C9	ω	GREEK SMALL LETTER OMEGA
U+03CA	ϊ	GREEK SMALL LETTER IOTA WITH DIALYTIKA
U+03CB	ϋ	GREEK SMALL LETTER UPSILON WITH DIALYTIKA
U+03CC	ό	GREEK SMALL LETTER OMICRON WITH TONOS
U+03CD	ύ	GREEK SMALL LETTER UPSILON WITH TONOS
U+03CE	ώ	GREEK SMALL LETTER OMEGA WITH TONOS
U+03D0	β	GREEK BETA SYMBOL
U+03D1	ϑ	GREEK THETA SYMBOL

Punto di codifica	Aspetto	Descrizione
U+03D2	ϒ	GREEK UPSILON WITH HOOK SYMBOL
U+03D3	□	GREEK UPSILON WITH ACUTE AND HOOK SYMBOL
U+03D4	□	GREEK UPSILON WITH DIAERESIS AND HOOK SYMBOL
U+03D5	ϕ	GREEK PHI SYMBOL
U+03D6	ϖ	GREEK PI SYMBOL
U+03D7	□	GREEK KAI SYMBOL
U+03D8	□	GREEK LETTER ARCHAIC KOPPA
U+03D9	□	GREEK SMALL LETTER ARCHAIC KOPPA
U+03DA	□	GREEK LETTER STIGMA
U+03DB	ϛ	GREEK SMALL LETTER STIGMA
U+03DC	Ϝ	GREEK LETTER DIGAMMA
U+03DD	ϝ	GREEK SMALL LETTER DIGAMMA
U+03DE	□	GREEK LETTER KOPPA
U+03DF	ϟ	GREEK SMALL LETTER KOPPA
U+03E0	□	GREEK LETTER SAMPI
U+03E1	ϡ	GREEK SMALL LETTER SAMPI
U+03E2	□	COPTIC CAPITAL LETTER SHEI
U+03E3	□	COPTIC SMALL LETTER SHEI
U+03E4	□	COPTIC CAPITAL LETTER FEI
U+03E5	□	COPTIC SMALL LETTER FEI
U+03E6	□	COPTIC CAPITAL LETTER KHEI
U+03E7	□	COPTIC SMALL LETTER KHEI
U+03E8	□	COPTIC CAPITAL LETTER HORI
U+03E9	□	COPTIC SMALL LETTER HORI
U+03EA	□	COPTIC CAPITAL LETTER GANGIA
U+03EB	□	COPTIC SMALL LETTER GANGIA
U+03EC	□	COPTIC CAPITAL LETTER SHIMA
U+03ED	□	COPTIC SMALL LETTER SHIMA
U+03EE	□	COPTIC CAPITAL LETTER DEI
U+03EF	□	COPTIC SMALL LETTER DEI
U+03F0	κ	GREEK KAPPA SYMBOL
U+03F1	ρ	GREEK RHO SYMBOL
U+03F2	ς	GREEK LUNATE SIGMA SYMBOL
U+03F3	□	GREEK LETTER YOT
U+03F4	Θ	GREEK CAPITAL THETA SYMBOL
U+03F5	ε	GREEK LUNATE EPSILON SYMBOL
U+03F6	□	GREEK REVERSED LUNATE EPSILON SYMBOL
U+03F7	□	GREEK CAPITAL LETTER SHO
U+03F8	□	GREEK SMALL LETTER SHO
U+03F9	Σ	GREEK CAPITAL LUNATE SIGMA SYMBOL
U+03FA	□	GREEK CAPITAL LETTER SAN
U+03FB	□	GREEK SMALL LETTER SAN

Tabella u80.5. *Cyrillic*.

Punto di codifica	Aspetto	Descrizione
U+0400	□	CYRILLIC CAPITAL LETTER IE WITH GRAVE
U+0401	Ё	CYRILLIC CAPITAL LETTER IO
U+0402	Ђ	CYRILLIC CAPITAL LETTER DJE (Serbocroatian)
U+0403	Ѓ	CYRILLIC CAPITAL LETTER GJE
U+0404	Є	CYRILLIC CAPITAL LETTER UKRAINIAN IE
U+0405	S	CYRILLIC CAPITAL LETTER DZE
U+0406	I	CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I
U+0407	□	CYRILLIC CAPITAL LETTER YI (Ukrainian)
U+0408	J	CYRILLIC CAPITAL LETTER JE
U+0409	Љ	CYRILLIC CAPITAL LETTER LJE
U+040A	Њ	CYRILLIC CAPITAL LETTER NJE
U+040B	Ћ	CYRILLIC CAPITAL LETTER TSHE (Serbocroatian)
U+040C	Ќ	CYRILLIC CAPITAL LETTER KJE
U+040D	□	CYRILLIC CAPITAL LETTER I WITH GRAVE
U+040E	Ў	CYRILLIC CAPITAL LETTER SHORT U (Byelorussian)
U+040F	Џ	CYRILLIC CAPITAL LETTER DZHE
U+0410	А	CYRILLIC CAPITAL LETTER A
U+0411	Б	CYRILLIC CAPITAL LETTER BE
U+0412	В	CYRILLIC CAPITAL LETTER VE
U+0413	Г	CYRILLIC CAPITAL LETTER GHE
U+0414	Д	CYRILLIC CAPITAL LETTER DE
U+0415	Е	CYRILLIC CAPITAL LETTER IE
U+0416	Ж	CYRILLIC CAPITAL LETTER ZHE
U+0417	З	CYRILLIC CAPITAL LETTER ZE
U+0418	И	CYRILLIC CAPITAL LETTER I
U+0419	Й	CYRILLIC CAPITAL LETTER SHORT I
U+041A	К	CYRILLIC CAPITAL LETTER KA
U+041B	Л	CYRILLIC CAPITAL LETTER EL
U+041C	М	CYRILLIC CAPITAL LETTER EM
U+041D	Н	CYRILLIC CAPITAL LETTER EN
U+041E	О	CYRILLIC CAPITAL LETTER O
U+041F	П	CYRILLIC CAPITAL LETTER PE
U+0420	Р	CYRILLIC CAPITAL LETTER ER
U+0421	С	CYRILLIC CAPITAL LETTER ES
U+0422	Т	CYRILLIC CAPITAL LETTER TE
U+0423	У	CYRILLIC CAPITAL LETTER U

Punto di codifica	Aspetto	Descrizione
U+0424	Ф	CYRILLIC CAPITAL LETTER EF
U+0425	Х	CYRILLIC CAPITAL LETTER HA
U+0426	Ц	CYRILLIC CAPITAL LETTER TSE
U+0427	Ч	CYRILLIC CAPITAL LETTER CHE
U+0428	Ш	CYRILLIC CAPITAL LETTER SHA
U+0429	Щ	CYRILLIC CAPITAL LETTER SHCHA
U+042A	Ъ	CYRILLIC CAPITAL LETTER HARD SIGN
U+042B	Ы	CYRILLIC CAPITAL LETTER YERU
U+042C	Ь	CYRILLIC CAPITAL LETTER SOFT SIGN
U+042D	Э	CYRILLIC CAPITAL LETTER E
U+042E	Ю	CYRILLIC CAPITAL LETTER YU
U+042F	Я	CYRILLIC CAPITAL LETTER YA
U+0430	а	CYRILLIC SMALL LETTER A
U+0431	б	CYRILLIC SMALL LETTER BE
U+0432	в	CYRILLIC SMALL LETTER VE
U+0433	г	CYRILLIC SMALL LETTER GHE
U+0434	д	CYRILLIC SMALL LETTER DE
U+0435	е	CYRILLIC SMALL LETTER IE
U+0436	ж	CYRILLIC SMALL LETTER ZHE
U+0437	з	CYRILLIC SMALL LETTER ZE
U+0438	и	CYRILLIC SMALL LETTER I
U+0439	й	CYRILLIC SMALL LETTER SHORT I
U+043A	к	CYRILLIC SMALL LETTER KA
U+043B	л	CYRILLIC SMALL LETTER EL
U+043C	м	CYRILLIC SMALL LETTER EM
U+043D	н	CYRILLIC SMALL LETTER EN
U+043E	о	CYRILLIC SMALL LETTER O
U+043F	п	CYRILLIC SMALL LETTER PE
U+0440	р	CYRILLIC SMALL LETTER ER
U+0441	с	CYRILLIC SMALL LETTER ES
U+0442	т	CYRILLIC SMALL LETTER TE
U+0443	у	CYRILLIC SMALL LETTER U
U+0444	ф	CYRILLIC SMALL LETTER EF
U+0445	х	CYRILLIC SMALL LETTER HA
U+0446	ц	CYRILLIC SMALL LETTER TSE
U+0447	ч	CYRILLIC SMALL LETTER CHE
U+0448	ш	CYRILLIC SMALL LETTER SHA
U+0449	щ	CYRILLIC SMALL LETTER SHCHA
U+044A	ъ	CYRILLIC SMALL LETTER HARD SIGN
U+044B	ы	CYRILLIC SMALL LETTER YERU
U+044C	ь	CYRILLIC SMALL LETTER SOFT SIGN
U+044D	э	CYRILLIC SMALL LETTER E
U+044E	ю	CYRILLIC SMALL LETTER YU
U+044F	я	CYRILLIC SMALL LETTER YA
U+0450	□	CYRILLIC SMALL LETTER IE WITH GRAVE
U+0451	ё	CYRILLIC SMALL LETTER IO
U+0452	ђ	CYRILLIC SMALL LETTER DJE (Serbocroatian)
U+0453	ѓ	CYRILLIC SMALL LETTER GJE
U+0454	є	CYRILLIC SMALL LETTER UKRAINIAN IE
U+0455	ѕ	CYRILLIC SMALL LETTER DZE
U+0456	і	CYRILLIC SMALL LETTER BYELORUSSIAN-UKRAINIAN I
U+0457	ї	CYRILLIC SMALL LETTER YI (Ukrainian)
U+0458	ј	CYRILLIC SMALL LETTER JE
U+0459	љ	CYRILLIC SMALL LETTER LJE
U+045A	њ	CYRILLIC SMALL LETTER NJE
U+045B	ћ	CYRILLIC SMALL LETTER TSHE (Serbocroatian)
U+045C	ќ	CYRILLIC SMALL LETTER KJE
U+045D	□	CYRILLIC SMALL LETTER I WITH GRAVE
U+045E	ў	CYRILLIC SMALL LETTER SHORT U (Byelorussian)
U+045F	џ	CYRILLIC SMALL LETTER DZHE
U+0460	□	CYRILLIC CAPITAL LETTER OMEGA
U+0461	□	CYRILLIC SMALL LETTER OMEGA
U+0462		CYRILLIC CAPITAL LETTER YAT
U+0463		CYRILLIC SMALL LETTER YAT
U+0464	□	CYRILLIC CAPITAL LETTER IOTIFIED E
U+0465	□	CYRILLIC SMALL LETTER IOTIFIED E
U+0466	□	CYRILLIC CAPITAL LETTER LITTLE YUS
U+0467	□	CYRILLIC SMALL LETTER LITTLE YUS
U+0468	□	CYRILLIC CAPITAL LETTER IOTIFIED LITTLE YUS
U+0469	□	CYRILLIC SMALL LETTER IOTIFIED LITTLE YUS
U+046A	□	CYRILLIC CAPITAL LETTER BIG YUS
U+046B	□	CYRILLIC SMALL LETTER BIG YUS
U+046C	□	CYRILLIC CAPITAL LETTER IOTIFIED BIG YUS
U+046D	□	CYRILLIC SMALL LETTER IOTIFIED BIG YUS
U+046E	□	CYRILLIC CAPITAL LETTER KSI
U+046F	□	CYRILLIC SMALL LETTER KSI
U+0470	□	CYRILLIC CAPITAL LETTER PSI
U+0471	□	CYRILLIC SMALL LETTER PSI
U+0472		CYRILLIC CAPITAL LETTER FITA
U+0473		CYRILLIC SMALL LETTER FITA
U+0474		CYRILLIC CAPITAL LETTER IZHITSA
U+0475		CYRILLIC SMALL LETTER IZHITSA
U+0476	□	CYRILLIC CAPITAL LETTER IZHITSA WITH DOUBLE GRAVE ACCENT
U+0477	□	CYRILLIC SMALL LETTER IZHITSA WITH DOUBLE GRAVE ACCENT
U+0478	□	CYRILLIC CAPITAL LETTER UK

Punto di codifica	Aspetto	Descrizione
U+0479	□	CYRILLIC SMALL LETTER UK
U+047A	□	CYRILLIC CAPITAL LETTER ROUND OMEGA
U+047B	□	CYRILLIC SMALL LETTER ROUND OMEGA
U+047C	□	CYRILLIC CAPITAL LETTER OMEGA WITH TITLO
U+047D	□	CYRILLIC SMALL LETTER OMEGA WITH TITLO
U+047E	□	CYRILLIC CAPITAL LETTER OT
U+047F	□	CYRILLIC SMALL LETTER OT
U+0480	□	CYRILLIC CAPITAL LETTER KOPPA
U+0481	□	CYRILLIC SMALL LETTER KOPPA
U+0482	□	CYRILLIC THOUSANDS SIGN
U+0483	□	COMBINING CYRILLIC TITLO
U+0484	□	COMBINING CYRILLIC PALATALIZATION
U+0485	□	COMBINING CYRILLIC DASIA PNEUMATA
U+0486	□	COMBINING CYRILLIC PSILI PNEUMATA
U+0488	□	COMBINING CYRILLIC HUNDRED THOUSANDS SIGN
U+0489	□	COMBINING CYRILLIC MILLIONS SIGN
U+048A	□	CYRILLIC CAPITAL LETTER SHORT I WITH TAIL
U+048B	□	CYRILLIC SMALL LETTER SHORT I WITH TAIL
U+048C	□	CYRILLIC CAPITAL LETTER SEMISOFT SIGN
U+048D	□	CYRILLIC SMALL LETTER SEMISOFT SIGN
U+048E	□	CYRILLIC CAPITAL LETTER ER WITH TICK
U+048F	□	CYRILLIC SMALL LETTER ER WITH TICK
U+0490	□	CYRILLIC CAPITAL LETTER GHE WITH UPTURN
U+0491	□	CYRILLIC SMALL LETTER GHE WITH UPTURN
U+0492	□	CYRILLIC CAPITAL LETTER GHE WITH STROKE
U+0493	□	CYRILLIC SMALL LETTER GHE WITH STROKE
U+0494	□	CYRILLIC CAPITAL LETTER GHE WITH MIDDLE HOOK
U+0495	□	CYRILLIC SMALL LETTER GHE WITH MIDDLE HOOK
U+0496	□	CYRILLIC CAPITAL LETTER ZHE WITH DESCENDER
U+0497	□	CYRILLIC SMALL LETTER ZHE WITH DESCENDER
U+0498	□	CYRILLIC CAPITAL LETTER ZE WITH DESCENDER
U+0499	□	CYRILLIC SMALL LETTER ZE WITH DESCENDER
U+049A	□	CYRILLIC CAPITAL LETTER KA WITH DESCENDER
U+049B	□	CYRILLIC SMALL LETTER KA WITH DESCENDER
U+049C	□	CYRILLIC CAPITAL LETTER KA WITH VERTICAL STROKE
U+049D	□	CYRILLIC SMALL LETTER KA WITH VERTICAL STROKE
U+049E	□	CYRILLIC CAPITAL LETTER KA WITH STROKE
U+049F	□	CYRILLIC SMALL LETTER KA WITH STROKE
U+04A0	□	CYRILLIC CAPITAL LETTER BASHKIR KA
U+04A1	□	CYRILLIC SMALL LETTER BASHKIR KA
U+04A2	□	CYRILLIC CAPITAL LETTER EN WITH DESCENDER
U+04A3	□	CYRILLIC SMALL LETTER EN WITH DESCENDER
U+04A4	□	CYRILLIC CAPITAL LIGATURE EN GHE
U+04A5	□	CYRILLIC SMALL LIGATURE EN GHE
U+04A6	□	CYRILLIC CAPITAL LETTER PE WITH MIDDLE HOOK (Abkhasian)
U+04A7	□	CYRILLIC SMALL LETTER PE WITH MIDDLE HOOK (Abkhasian)
U+04A8	□	CYRILLIC CAPITAL LETTER ABKHASIAN HA
U+04A9	□	CYRILLIC SMALL LETTER ABKHASIAN HA
U+04AA	□	CYRILLIC CAPITAL LETTER ES WITH DESCENDER
U+04AB	□	CYRILLIC SMALL LETTER ES WITH DESCENDER
U+04AC	□	CYRILLIC CAPITAL LETTER TE WITH DESCENDER
U+04AD	□	CYRILLIC SMALL LETTER TE WITH DESCENDER
U+04AE	□	CYRILLIC CAPITAL LETTER STRAIGHT U
U+04AF	□	CYRILLIC SMALL LETTER STRAIGHT U
U+04B0	□	CYRILLIC CAPITAL LETTER STRAIGHT U WITH STROKE
U+04B1	□	CYRILLIC SMALL LETTER STRAIGHT U WITH STROKE
U+04B2	□	CYRILLIC CAPITAL LETTER HA WITH DESCENDER
U+04B3	□	CYRILLIC SMALL LETTER HA WITH DESCENDER
U+04B4	□	CYRILLIC CAPITAL LIGATURE TE TSE (Abkhasian)
U+04B5	□	CYRILLIC SMALL LIGATURE TE TSE (Abkhasian)
U+04B6	□	CYRILLIC CAPITAL LETTER CHE WITH DESCENDER
U+04B7	□	CYRILLIC SMALL LETTER CHE WITH DESCENDER
U+04B8	□	CYRILLIC CAPITAL LETTER CHE WITH VERTICAL STROKE
U+04B9	□	CYRILLIC SMALL LETTER CHE WITH VERTICAL STROKE
U+04BA	□	CYRILLIC CAPITAL LETTER SHHA
U+04BB	□	CYRILLIC SMALL LETTER SHHA
U+04BC	□	CYRILLIC CAPITAL LETTER ABKHASIAN CHE
U+04BD	□	CYRILLIC SMALL LETTER ABKHASIAN CHE
U+04BE	□	CYRILLIC CAPITAL LETTER ABKHASIAN CHE WITH DESCENDER
U+04BF	□	CYRILLIC SMALL LETTER ABKHASIAN CHE WITH DESCENDER
U+04C0	□	CYRILLIC LETTER PALOCHKA
U+04C1	Ӂ	CYRILLIC CAPITAL LETTER ZHE WITH BREVE
U+04C2	ӂ	CYRILLIC SMALL LETTER ZHE WITH BREVE
U+04C3	□	CYRILLIC CAPITAL LETTER KA WITH HOOK
U+04C4	□	CYRILLIC SMALL LETTER KA WITH HOOK
U+04C5	□	CYRILLIC CAPITAL LETTER EL WITH TAIL
U+04C6	□	CYRILLIC SMALL LETTER EL WITH TAIL
U+04C7	□	CYRILLIC CAPITAL LETTER EN WITH HOOK
U+04C8	□	CYRILLIC SMALL LETTER EN WITH HOOK

Punto di codifica	Aspetto	Descrizione
U+04C9	□	CYRILLIC CAPITAL LETTER EN WITH TAIL
U+04CA	□	CYRILLIC SMALL LETTER EN WITH TAIL
U+04CB	□	CYRILLIC CAPITAL LETTER KHAKASSIAN CHE
U+04CC	□	CYRILLIC SMALL LETTER KHAKASSIAN CHE
U+04CD	□	CYRILLIC CAPITAL LETTER EM WITH TAIL
U+04CE	□	CYRILLIC SMALL LETTER EM WITH TAIL
U+04D0	Ӑ	CYRILLIC CAPITAL LETTER A WITH BREVE
U+04D1	ӑ	CYRILLIC SMALL LETTER A WITH BREVE
U+04D2	Ӓ	CYRILLIC CAPITAL LETTER A WITH DIAERESIS
U+04D3	ӓ	CYRILLIC SMALL LETTER A WITH DIAERESIS
U+04D4	□	CYRILLIC CAPITAL LIGATURE A IE
U+04D5	□	CYRILLIC SMALL LIGATURE A IE
U+04D6	Ӗ	CYRILLIC CAPITAL LETTER IE WITH BREVE
U+04D7	ӗ	CYRILLIC SMALL LETTER IE WITH BREVE
U+04D8	□	CYRILLIC CAPITAL LETTER SCHWA
U+04D9	□	CYRILLIC SMALL LETTER SCHWA
U+04DA	□	CYRILLIC CAPITAL LETTER SCHWA WITH DIAERESIS
U+04DB	□	CYRILLIC SMALL LETTER SCHWA WITH DIAERESIS
U+04DC	Ӝ	CYRILLIC CAPITAL LETTER ZHE WITH DIAERESIS
U+04DD	ӝ	CYRILLIC SMALL LETTER ZHE WITH DIAERESIS
U+04DE	Ӟ	CYRILLIC CAPITAL LETTER ZE WITH DIAERESIS
U+04DF	ӟ	CYRILLIC SMALL LETTER ZE WITH DIAERESIS
U+04E0	□	CYRILLIC CAPITAL LETTER ABKHASIAN DZE
U+04E1	□	CYRILLIC SMALL LETTER ABKHASIAN DZE
U+04E2	□	CYRILLIC CAPITAL LETTER I WITH MACRON
U+04E3	□	CYRILLIC SMALL LETTER I WITH MACRON
U+04E4	□	CYRILLIC CAPITAL LETTER I WITH DIAERESIS
U+04E5	□	CYRILLIC SMALL LETTER I WITH DIAERESIS
U+04E6	□	CYRILLIC CAPITAL LETTER O WITH DIAERESIS
U+04E7	□	CYRILLIC SMALL LETTER O WITH DIAERESIS
U+04E8	□	CYRILLIC CAPITAL LETTER BARRED O
U+04E9	□	CYRILLIC SMALL LETTER BARRED O
U+04EA	□	CYRILLIC CAPITAL LETTER BARRED O WITH DIAERESIS
U+04EB	□	CYRILLIC SMALL LETTER BARRED O WITH DIAERESIS
U+04EC	□	CYRILLIC CAPITAL LETTER E WITH DIAERESIS
U+04ED	□	CYRILLIC SMALL LETTER E WITH DIAERESIS
U+04EE	□	CYRILLIC CAPITAL LETTER U WITH MACRON
U+04EF	□	CYRILLIC SMALL LETTER U WITH MACRON
U+04F0	□	CYRILLIC CAPITAL LETTER U WITH DIAERESIS
U+04F1	□	CYRILLIC SMALL LETTER U WITH DIAERESIS
U+04F2	□	CYRILLIC CAPITAL LETTER U WITH DOUBLE ACUTE
U+04F3	□	CYRILLIC SMALL LETTER U WITH DOUBLE ACUTE
U+04F4	□	CYRILLIC CAPITAL LETTER CHE WITH DIAERESIS
U+04F5	□	CYRILLIC SMALL LETTER CHE WITH DIAERESIS
U+04F8	□	CYRILLIC CAPITAL LETTER YERU WITH DIAERESIS
U+04F9	□	CYRILLIC SMALL LETTER YERU WITH DIAERESIS

Tabella u80.6. *General punctuation.*

Punto di codifica	Aspetto	Descrizione
U+2000	□	EN QUAD
U+2001	□	EM QUAD
U+2002	□	EN SPACE
U+2003	□	EM SPACE
U+2004	□	THREE-PER-EM SPACE
U+2005	□	FOUR-PER-EM SPACE
U+2006	□	SIX-PER-EM SPACE
U+2007	□	FIGURE SPACE
U+2008	□	PUNCTUATION SPACE
U+2009	□	THIN SPACE
U+200A	□	HAIR SPACE
U+200B	□	ZERO WIDTH SPACE
U+200C	□	ZERO WIDTH NON-JOINER
U+200D	□	ZERO WIDTH JOINER
U+200E	□	LEFT-TO-RIGHT MARK
U+200F	□	RIGHT-TO-LEFT MARK
U+2010	-	HYPHEN
U+2011	□	NON-BREAKING HYPHEN
U+2012	□	FIGURE DASH
U+2013	–	EN DASH
U+2014	—	EM DASH
U+2015	□	HORIZONTAL BAR
U+2016	□	DOUBLE VERTICAL LINE
U+2017	□	DOUBLE LOW LINE
U+2018	‘	LEFT SINGLE QUOTATION MARK
U+2019	’	RIGHT SINGLE QUOTATION MARK
U+201A	’	SINGLE LOW-9 QUOTATION MARK
U+201B	□	SINGLE HIGH-REVERSED-9 QUOTATION MARK
U+201C	“	LEFT DOUBLE QUOTATION MARK
U+201D	”	RIGHT DOUBLE QUOTATION MARK
U+201E	”	DOUBLE LOW-9 QUOTATION MARK

Punto di codifica	Aspetto	Descrizione
U+201F	□	DOUBLE HIGH-REVERSED-9 QUOTATION MARK
U+2020	†	DAGGER
U+2021	‡	DOUBLE DAGGER
U+2022	•	BULLET
U+2023	□	TRIANGULAR BULLET
U+2024	‧	ONE DOT LEADER
U+2025	‥	TWO DOT LEADER
U+2026	…	HORIZONTAL ELLIPSIS
U+2027	□	HYPHENATION POINT
U+2028	□	LINE SEPARATOR
U+2029	□	PARAGRAPH SEPARATOR
U+202A	□	LEFT-TO-RIGHT EMBEDDING
U+202B	□	RIGHT-TO-LEFT EMBEDDING
U+202C	□	POP DIRECTIONAL FORMATTING
U+202D	□	LEFT-TO-RIGHT OVERRIDE
U+202E	□	RIGHT-TO-LEFT OVERRIDE
U+202F	□	NARROW NO-BREAK SPACE
U+2030	‰	PER MILLE SIGN
U+2031	‱	PER TEN THOUSAND SIGN
U+2032	′	PRIME
U+2033	″	DOUBLE PRIME
U+2034	‴	TRIPLE PRIME
U+2035	□	REVERSED PRIME
U+2036	□	REVERSED DOUBLE PRIME
U+2037	□	REVERSED TRIPLE PRIME
U+2038	□	CARET
U+2039	‹	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
U+203A	›	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
U+203B	※	REFERENCE MARK
U+203C	!!	DOUBLE EXCLAMATION MARK
U+203D	⁉	INTERROBANG
U+203E	‾	OVERLINE
U+203F	□	UNDERTIE (Enotikon)
U+2040	□	CHARACTER TIE
U+2041	□	CARET INSERTION POINT
U+2042	□	ASTERISM
U+2043	□	HYPHEN BULLET
U+2044	/	FRACTION SLASH
U+2045	□	LEFT SQUARE BRACKET WITH QUILL
U+2046	□	RIGHT SQUARE BRACKET WITH QUILL
U+2047	??	DOUBLE QUESTION MARK
U+2048	?!	QUESTION EXCLAMATION MARK
U+2049	!?	EXCLAMATION QUESTION MARK
U+204A	□	TIRONIAN SIGN ET
U+204B	□	REVERSED PILCROW SIGN
U+204C	□	BLACK LEFTWARDS BULLET
U+204D	□	BLACK RIGHTWARDS BULLET
U+204E	□	LOW ASTERISK
U+204F	□	REVERSED SEMICOLON
U+2050	□	CLOSE UP
U+2051	□	TWO ASTERISKS ALIGNED VERTICALLY
U+2052	□	COMMERCIAL MINUS SIGN
U+2053	□	SWUNG DASH
U+2054	□	INVERTED UNDERTIE
U+2057	⁗	QUADRUPLE PRIME
U+205F	□	MEDIUM MATHEMATICAL SPACE
U+2060	□	WORD JOINER
U+2061	□	FUNCTION APPLICATION
U+2062	□	INVISIBLE TIMES
U+2063	□	INVISIBLE SEPARATOR
U+206A	□	INHIBIT SYMMETRIC SWAPPING
U+206B	□	ACTIVATE SYMMETRIC SWAPPING
U+206C	□	INHIBIT ARABIC FORM SHAPING
U+206D	□	ACTIVATE ARABIC FORM SHAPING
U+206E	□	NATIONAL DIGIT SHAPES
U+206F	□	NOMINAL DIGIT SHAPES

Tabella u80.7. *Superscripts and subscripts.*

Punto di codifica	Aspetto	Descrizione
U+2070	⁰	SUPERSCRIPT ZERO
U+2071	ⁱ	SUPERSCRIPT LATIN SMALL LETTER I
U+2072		
U+2073		
U+2074	⁴	SUPERSCRIPT FOUR
U+2075	⁵	SUPERSCRIPT FIVE
U+2076	⁶	SUPERSCRIPT SIX
U+2077	⁷	SUPERSCRIPT SEVEN
U+2078	⁸	SUPERSCRIPT EIGHT
U+2079	⁹	SUPERSCRIPT NINE
U+207A	⁺	SUPERSCRIPT PLUS SIGN
U+207B	⁻	SUPERSCRIPT MINUS
U+207C	⁼	SUPERSCRIPT EQUALS SIGN
U+207D	⁽	SUPERSCRIPT LEFT PARENTHESIS
U+207E	⁾	SUPERSCRIPT RIGHT PARENTHESIS

Punto di codifica	Aspetto	Descrizione
U+207F	ⁿ	SUPERSCRIPT LATIN SMALL LETTER N
U+2080	₀	SUBSCRIPT ZERO
U+2081	₁	SUBSCRIPT ONE
U+2082	₂	SUBSCRIPT TWO
U+2083	₃	SUBSCRIPT THREE
U+2084	₄	SUBSCRIPT FOUR
U+2085	₅	SUBSCRIPT FIVE
U+2086	₆	SUBSCRIPT SIX
U+2087	₇	SUBSCRIPT SEVEN
U+2088	₈	SUBSCRIPT EIGHT
U+2089	₉	SUBSCRIPT NINE
U+208A	₊	SUBSCRIPT PLUS SIGN
U+208B	₋	SUBSCRIPT MINUS
U+208C	₌	SUBSCRIPT EQUALS SIGN
U+208D	₍	SUBSCRIPT LEFT PARENTHESIS
U+208E	₎	SUBSCRIPT RIGHT PARENTHESIS

Tabella u80.8. *Currency symbols.*

Punto di codifica	Aspetto	Descrizione
U+20A0	□	EURO-CURRENCY SIGN
U+20A1	₡	COLON SIGN
U+20A2	□	CRUZEIRO SIGN
U+20A3	□	FRENCH FRANC SIGN
U+20A4	£	LIRA SIGN
U+20A5	□	MILL SIGN
U+20A6	₦	NAIRA SIGN
U+20A7	□	PESETA SIGN
U+20A8	Rs	RUPEE SIGN
U+20A9	₩	WON SIGN
U+20AA	□	NEW SHEQEL SIGN
U+20AB	₫	DONG SIGN
U+20AC	€	EURO SIGN
U+20AD	□	KIP SIGN
U+20AE	□	TUGRIK SIGN
U+20AF	□	DRACHMA SIGN
U+20B0	□	GERMAN PENNY SIGN
U+20B1	□	PESO SIGN

Tabella u80.9. *Letterlike symbols.*

Punto di codifica	Aspetto	Descrizione
U+2100	a/c	ACCOUNT OF
U+2101	a/s	ADDRESSED TO THE SUBJECT
U+2102	□	DOUBLE-STRUCK CAPITAL C
U+2103	°C	DEGREE CELSIUS
U+2104	□	CENTRE LINE SYMBOL
U+2105	c/o	CARE OF
U+2106	c/u	U+CADA UNA
U+2107	□	EULER CONSTANT
U+2108	□	SCRUPLE
U+2109	°F	DEGREE FAHRENHEIT
U+210A	g	SCRIPT SMALL G
U+210B	ℋ	SCRIPT CAPITAL H
U+210C	ℌ	BLACK-LETTER CAPITAL H
U+210D	□	DOUBLE-STRUCK CAPITAL H
U+210E	ℎ	PLANCK CONSTANT
U+210F	ℏ	PLANCK CONSTANT OVER TWO PI
U+2110	ℐ	SCRIPT CAPITAL I
U+2111	ℑ	BLACK-LETTER CAPITAL I
U+2112	ℒ	SCRIPT CAPITAL L
U+2113	ℓ	SCRIPT SMALL L
U+2114	□	L B BAR SYMBOL
U+2115	□	DOUBLE-STRUCK CAPITAL N
U+2116	№	NUMERO SIGN
U+2117	□	SOUND RECORDING COPYRIGHT
U+2118	℘	SCRIPT CAPITAL P
U+2119	□	DOUBLE-STRUCK CAPITAL P
U+211A	□	DOUBLE-STRUCK CAPITAL Q
U+211B	ℛ	SCRIPT CAPITAL R
U+211C	ℜ	BLACK-LETTER CAPITAL R
U+211D	□	DOUBLE-STRUCK CAPITAL R
U+211E	□	PRESCRIPTION TAKE
U+211F	□	RESPONSE
U+2120	SM	SERVICE MARK
U+2121	TEL	TELEPHONE SIGN
U+2122	™	TRADE MARK SIGN
U+2123	□	VERSICLE
U+2124	□	DOUBLE-STRUCK CAPITAL Z
U+2125	□	OUNCE SIGN
U+2126	Ω	OHM SIGN
U+2127	℧	INVERTED OHM SIGN

Punto di codifica	Aspetto	Descrizione
U+2128	ℨ	BLACK-LETTER CAPITAL Z
U+2129	□	TURNED GREEK SMALL LETTER IOTA
U+212A	K	KELVIN SIGN
U+212B	Å	ANGSTROM SIGN
U+212C	ℬ	SCRIPT CAPITAL B
U+212D	ℭ	BLACK-LETTER CAPITAL C
U+212E	□	ESTIMATED SYMBOL
U+212F	e	SCRIPT SMALL E
U+2130	ℰ	SCRIPT CAPITAL E
U+2131	ℱ	SCRIPT CAPITAL F
U+2132	□	TURNED CAPITAL F
U+2133	ℳ	SCRIPT CAPITAL M
U+2134	ℴ	SCRIPT SMALL O
U+2135	א	ALEF SYMBOL
U+2136	ב	BET SYMBOL
U+2137	ג	GIMEL SYMBOL
U+2138	ד	DALET SYMBOL
U+2139	i	INFORMATION SOURCE
U+213A	□	ROTATED CAPITAL Q
U+213B	FAX	FACSIMILE SIGN
U+213D	□	DOUBLE-STRUCK SMALL GAMMA
U+213E	□	DOUBLE-STRUCK CAPITAL GAMMA
U+213F	□	DOUBLE-STRUCK CAPITAL PI
U+2140	∑	DOUBLE-STRUCK N-ARY SUMMATION
U+2141	□	TURNED SANS-SERIF CAPITAL G
U+2142	□	TURNED SANS-SERIF CAPITAL L
U+2143	□	REVERSED SANS-SERIF CAPITAL L
U+2144	□	TURNED SANS-SERIF CAPITAL Y
U+2145	D	DOUBLE-STRUCK ITALIC CAPITAL D
U+2146	d	DOUBLE-STRUCK ITALIC SMALL D
U+2147	e	DOUBLE-STRUCK ITALIC SMALL E
U+2148	i	DOUBLE-STRUCK ITALIC SMALL I
U+2149	j	DOUBLE-STRUCK ITALIC SMALL J
U+214A	□	PROPERTY LINE
U+214B	□	TURNED AMPERSAND

Tabella u80.10. *Number forms.*

Punto di codifica	Aspetto	Descrizione
U+2153	1/3	VULGAR FRACTION ONE THIRD
U+2154	2/3	VULGAR FRACTION TWO THIRDS
U+2155	1/5	VULGAR FRACTION ONE FIFTH
U+2156	2/5	VULGAR FRACTION TWO FIFTHS
U+2157	3/5	VULGAR FRACTION THREE FIFTHS
U+2158	4/5	VULGAR FRACTION FOUR FIFTHS
U+2159	1/6	VULGAR FRACTION ONE SIXTH
U+215A	5/6	VULGAR FRACTION FIVE SIXTHS
U+215B	1/8	VULGAR FRACTION ONE EIGHTH
U+215C	3/8	VULGAR FRACTION THREE EIGHTHS
U+215D	5/8	VULGAR FRACTION FIVE EIGHTHS
U+215E	7/8	VULGAR FRACTION SEVEN EIGHTHS
U+215F	1/	FRACTION NUMERATOR ONE
U+2160	I	ROMAN NUMERAL ONE
U+2161	II	ROMAN NUMERAL TWO
U+2162	III	ROMAN NUMERAL THREE
U+2163	IV	ROMAN NUMERAL FOUR
U+2164	V	ROMAN NUMERAL FIVE
U+2165	VI	ROMAN NUMERAL SIX
U+2166	VII	ROMAN NUMERAL SEVEN
U+2167	VIII	ROMAN NUMERAL EIGHT
U+2168	IX	ROMAN NUMERAL NINE
U+2169	X	ROMAN NUMERAL TEN
U+216A	XI	ROMAN NUMERAL ELEVEN
U+216B	XII	ROMAN NUMERAL TWELVE
U+216C	L	ROMAN NUMERAL FIFTY
U+216D	C	ROMAN NUMERAL ONE HUNDRED
U+216E	D	ROMAN NUMERAL FIVE HUNDRED
U+216F	M	ROMAN NUMERAL ONE THOUSAND
U+2170	i	SMALL ROMAN NUMERAL ONE
U+2171	ii	SMALL ROMAN NUMERAL TWO
U+2172	iii	SMALL ROMAN NUMERAL THREE
U+2173	iv	SMALL ROMAN NUMERAL FOUR
U+2174	v	SMALL ROMAN NUMERAL FIVE
U+2175	vi	SMALL ROMAN NUMERAL SIX
U+2176	vii	SMALL ROMAN NUMERAL SEVEN
U+2177	viii	SMALL ROMAN NUMERAL EIGHT
U+2178	ix	SMALL ROMAN NUMERAL NINE
U+2179	x	SMALL ROMAN NUMERAL TEN
U+217A	xi	SMALL ROMAN NUMERAL ELEVEN
U+217B	xii	SMALL ROMAN NUMERAL TWELVE
U+217C	l	SMALL ROMAN NUMERAL FIFTY
U+217D	c	SMALL ROMAN NUMERAL ONE HUNDRED
U+217E	d	SMALL ROMAN NUMERAL FIVE HUNDRED
U+217F	m	SMALL ROMAN NUMERAL ONE THOUSAND
U+2180	□	ROMAN NUMERAL ONE THOUSAND C D

Punto di codifica	Aspetto	Descrizione
U+2181	□	ROMAN NUMERAL FIVE THOUSAND
U+2182	□	ROMAN NUMERAL TEN THOUSAND
U+2183	□	ROMAN NUMERAL REVERSED ONE HUNDRED

Tabella u80.11. *Arrows.*

Punto di codifica	Aspetto	Descrizione
U+2190	←	LEFTWARDS ARROW
U+2191	↑	UPWARDS ARROW
U+2192	→	RIGHTWARDS ARROW
U+2193	v	DOWNWARDS ARROW
U+2194	↔	LEFT RIGHT ARROW
U+2195	↕	UP DOWN ARROW
U+2196	□	NORTH WEST ARROW
U+2197	□	NORTH EAST ARROW
U+2198	□	SOUTH EAST ARROW
U+2199	□	SOUTH WEST ARROW
U+219A	□	LEFTWARDS ARROW WITH STROKE
U+219B	□	RIGHTWARDS ARROW WITH STROKE
U+219C	□	LEFTWARDS WAVE ARROW
U+219D	⇝	RIGHTWARDS WAVE ARROW
U+219E	□	LEFTWARDS TWO HEADED ARROW
U+219F	□	UPWARDS TWO HEADED ARROW
U+21A0	□	RIGHTWARDS TWO HEADED ARROW
U+21A1	□	DOWNWARDS TWO HEADED ARROW
U+21A2	□	LEFTWARDS ARROW WITH TAIL
U+21A3	□	RIGHTWARDS ARROW WITH TAIL
U+21A4	□	LEFTWARDS ARROW FROM BAR
U+21A5	□	UPWARDS ARROW FROM BAR
U+21A6	□	RIGHTWARDS ARROW FROM BAR
U+21A7	□	DOWNWARDS ARROW FROM BAR
U+21A8	□	UP DOWN ARROW WITH BASE
U+21A9	□	LEFTWARDS ARROW WITH HOOK
U+21AA	□	RIGHTWARDS ARROW WITH HOOK
U+21AB	□	LEFTWARDS ARROW WITH LOOP
U+21AC	□	RIGHTWARDS ARROW WITH LOOP
U+21AD	□	LEFT RIGHT WAVE ARROW
U+21AE	□	LEFT RIGHT ARROW WITH STROKE
U+21AF	□	DOWNWARDS ZIGZAG ARROW
U+21B0	□	UPWARDS ARROW WITH TIP LEFTWARDS
U+21B1	□	UPWARDS ARROW WITH TIP RIGHTWARDS
U+21B2	□	DOWNWARDS ARROW WITH TIP LEFTWARDS
U+21B3	□	DOWNWARDS ARROW WITH TIP RIGHTWARDS
U+21B4	□	RIGHTWARDS ARROW WITH CORNER DOWNWARDS
U+21B5	↵	DOWNWARDS ARROW WITH CORNER LEFTWARDS
U+21B6	□	ANTICLOCKWISE TOP SEMICIRCLE ARROW
U+21B7	□	CLOCKWISE TOP SEMICIRCLE ARROW
U+21B8	□	NORTH WEST ARROW TO LONG BAR
U+21B9	□	LEFTWARDS ARROW TO BAR OVER RIGHTWARDS ARROW TO BAR
U+21BA	□	ANTICLOCKWISE OPEN CIRCLE ARROW
U+21BB	□	CLOCKWISE OPEN CIRCLE ARROW
U+21BC	□	LEFTWARDS HARPOON WITH BARB UPWARDS
U+21BD	□	LEFTWARDS HARPOON WITH BARB DOWNWARDS
U+21BE	□	UPWARDS HARPOON WITH BARB RIGHTWARDS
U+21BF	□	UPWARDS HARPOON WITH BARB LEFTWARDS
U+21C0	□	RIGHTWARDS HARPOON WITH BARB UPWARDS
U+21C1	□	RIGHTWARDS HARPOON WITH BARB DOWNWARDS
U+21C2	□	DOWNWARDS HARPOON WITH BARB RIGHTWARDS
U+21C3	□	DOWNWARDS HARPOON WITH BARB LEFTWARDS
U+21C4	□	RIGHTWARDS ARROW OVER LEFTWARDS ARROW
U+21C5	□	UPWARDS ARROW LEFTWARDS OF DOWNWARDS ARROW
U+21C6	□	LEFTWARDS ARROW OVER RIGHTWARDS ARROW
U+21C7	□	LEFTWARDS PAIRED ARROWS
U+21C8	□	UPWARDS PAIRED ARROWS
U+21C9	□	RIGHTWARDS PAIRED ARROWS
U+21CA	□	DOWNWARDS PAIRED ARROWS
U+21CB	□	LEFTWARDS HARPOON OVER RIGHTWARDS HARPOON
U+21CC	⇌	RIGHTWARDS HARPOON OVER LEFTWARDS HARPOON
U+21CD	□	LEFTWARDS DOUBLE ARROW WITH STROKE
U+21CE	□	LEFT RIGHT DOUBLE ARROW WITH STROKE
U+21CF	□	RIGHTWARDS DOUBLE ARROW WITH STROKE
U+21D0	⇐	LEFTWARDS DOUBLE ARROW
U+21D1	⇑	UPWARDS DOUBLE ARROW
U+21D2	⇒	RIGHTWARDS DOUBLE ARROW
U+21D3	⇓	DOWNWARDS DOUBLE ARROW
U+21D4	⇔	LEFT RIGHT DOUBLE ARROW
U+21D5	□	UP DOWN DOUBLE ARROW
U+21D6	□	NORTH WEST DOUBLE ARROW

Punto di codifica	Aspetto	Descrizione
U+21D7	□	NORTH EAST DOUBLE ARROW
U+21D8	□	SOUTH EAST DOUBLE ARROW
U+21D9	□	SOUTH WEST DOUBLE ARROW
U+21DA	□	LEFTWARDS TRIPLE ARROW
U+21DB	□	RIGHTWARDS TRIPLE ARROW
U+21DC	□	LEFTWARDS SQUIGGLE ARROW
U+21DD	□	RIGHTWARDS SQUIGGLE ARROW
U+21DE	□	UPWARDS ARROW WITH DOUBLE STROKE
U+21DF	□	DOWNWARDS ARROW WITH DOUBLE STROKE
U+21E0	□	LEFTWARDS DASHED ARROW
U+21E1	□	UPWARDS DASHED ARROW
U+21E2	□	RIGHTWARDS DASHED ARROW
U+21E3	□	DOWNWARDS DASHED ARROW
U+21E4	□	LEFTWARDS ARROW TO BAR
U+21E5	□	RIGHTWARDS ARROW TO BAR
U+21E6	□	LEFTWARDS WHITE ARROW
U+21E7	□	UPWARDS WHITE ARROW
U+21E8	□	RIGHTWARDS WHITE ARROW
U+21E9	□	DOWNWARDS WHITE ARROW
U+21EA	□	UPWARDS WHITE ARROW FROM BAR
U+21EB	□	UPWARDS WHITE ARROW ON PEDESTAL
U+21EC	□	UPWARDS WHITE ARROW ON PEDESTAL WITH HORIZONTAL BAR
U+21ED	□	UPWARDS WHITE ARROW ON PEDESTAL WITH VERTICAL BAR
U+21EE	□	UPWARDS WHITE DOUBLE ARROW
U+21EF	□	UPWARDS WHITE DOUBLE ARROW ON PEDESTAL
U+21F0	□	RIGHTWARDS WHITE ARROW FROM WALL
U+21F1	□	NORTH WEST ARROW TO CORNER
U+21F2	□	SOUTH EAST ARROW TO CORNER
U+21F3	□	UP DOWN WHITE ARROW
U+21F4	□	RIGHT ARROW WITH SMALL CIRCLE
U+21F5	□	DOWNWARDS ARROW LEFTWARDS OF UPWARDS ARROW
U+21F6	□	THREE RIGHTWARDS ARROWS
U+21F7	□	LEFTWARDS ARROW WITH VERTICAL STROKE
U+21F8	□	RIGHTWARDS ARROW WITH VERTICAL STROKE
U+21F9	□	LEFT RIGHT ARROW WITH VERTICAL STROKE
U+21FA	□	LEFTWARDS ARROW WITH DOUBLE VERTICAL STROKE
U+21FB	□	RIGHTWARDS ARROW WITH DOUBLE VERTICAL STROKE
U+21FC	□	LEFT RIGHT ARROW WITH DOUBLE VERTICAL STROKE
U+21FD	□	LEFTWARDS OPEN-HEADED ARROW
U+21FE	□	RIGHTWARDS OPEN-HEADED ARROW
U+21FF	□	LEFT RIGHT OPEN-HEADED ARROW

Tabella u80.12. *Mathematical operators.*

Punto di codifica	Aspetto	Descrizione
U+2200	∀	FOR ALL
U+2201	∁	COMPLEMENT
U+2202	∂	PARTIAL DIFFERENTIAL
U+2203	∃	THERE EXISTS
U+2204	∄	THERE DOES NOT EXIST
U+2205	∅	EMPTY SET
U+2206	∆	INCREMENT
U+2207	∇	NABLA
U+2208	∈	ELEMENT OF
U+2209	∉	NOT AN ELEMENT OF
U+220A	□	SMALL ELEMENT OF
U+220B	∋	CONTAINS AS MEMBER
U+220C	∌	DOES NOT CONTAIN AS MEMBER
U+220D	□	SMALL CONTAINS AS MEMBER
U+220E	□	END OF PROOF
U+220F	∏	N-ARY PRODUCT
U+2210	∐	N-ARY COPRODUCT
U+2211	∑	N-ARY SUMMATION
U+2212	−	MINUS SIGN
U+2213	∓	MINUS-OR-PLUS SIGN
U+2214	∔	DOT PLUS
U+2215	∕	DIVISION SLASH
U+2216	∖	SET MINUS
U+2217	∗	ASTERISK OPERATOR
U+2218	∘	RING OPERATOR
U+2219	∙	BULLET OPERATOR
U+221A	√	SQUARE ROOT
U+221B	□	CUBE ROOT
U+221C	□	FOURTH ROOT
U+221D	∝	PROPORTIONAL TO
U+221E	∞	INFINITY
U+221F	□	RIGHT ANGLE
U+2220	∠	ANGLE
U+2221	∡	MEASURED ANGLE
U+2222	∢	SPHERICAL ANGLE
U+2223	∣	DIVIDES

Punto di codifica	Aspetto	Descrizione
U+2224	∤	DOES NOT DIVIDE
U+2225	∥	PARALLEL TO
U+2226	∦	NOT PARALLEL TO
U+2227	∧	LOGICAL AND
U+2228	∨	LOGICAL OR
U+2229	∩	INTERSECTION
U+222A	∪	UNION
U+222B	∫	INTEGRAL
U+222C	∬	DOUBLE INTEGRAL
U+222D	∭	TRIPLE INTEGRAL
U+222E	∮	CONTOUR INTEGRAL
U+222F	∯	SURFACE INTEGRAL
U+2230	∰	VOLUME INTEGRAL
U+2231	□	CLOCKWISE INTEGRAL
U+2232	□	CLOCKWISE CONTOUR INTEGRAL
U+2233	□	ANTICLOCKWISE CONTOUR INTEGRAL
U+2234	∴	THEREFORE
U+2235	∵	U+BECAUSE
U+2236	∶	RATIO
U+2237	□	PROPORTION
U+2238	□	DOT MINUS
U+2239	□	EXCESS
U+223A	□	GEOMETRIC PROPORTION
U+223B	□	HOMOTHETIC
U+223C	∼	TILDE OPERATOR
U+223D	∽	REVERSED TILDE (lazy S)
U+223E	□	INVERTED LAZY S
U+223F	□	SINE WAVE
U+2240	≀	WREATH PRODUCT
U+2241	≁	NOT TILDE
U+2242	□	MINUS TILDE
U+2243	≃	ASYMPTOTICALLY EQUAL TO
U+2244	≄	NOT ASYMPTOTICALLY EQUAL TO
U+2245	≅	APPROXIMATELY EQUAL TO
U+2246	□	APPROXIMATELY BUT NOT ACTUALLY EQUAL TO
U+2247	≇	NEITHER APPROXIMATELY NOR ACTUALLY EQUAL TO
U+2248	≈	ALMOST EQUAL TO
U+2249	≉	NOT ALMOST EQUAL TO
U+224A	≊	ALMOST EQUAL OR EQUAL TO
U+224B	□	TRIPLE TILDE
U+224C	□	ALL EQUAL TO
U+224D	≍	EQUIVALENT TO
U+224E	≎	GEOMETRICALLY EQUIVALENT TO
U+224F	≏	DIFFERENCE BETWEEN
U+2250	≐	APPROACHES THE LIMIT
U+2251	≑	GEOMETRICALLY EQUAL TO
U+2252	≒	APPROXIMATELY EQUAL TO OR THE IMAGE OF
U+2253	≓	IMAGE OF OR APPROXIMATELY EQUAL TO
U+2254	□	COLON EQUALS
U+2255	□	EQUALS COLON
U+2256	≖	RING IN EQUAL TO
U+2257	≗	RING EQUAL TO
U+2258	□	CORRESPONDS TO
U+2259	□	ESTIMATES
U+225A	□	EQUIANGULAR TO
U+225B	□	STAR EQUALS
U+225C	≜	DELTA EQUAL TO
U+225D	□	EQUAL TO BY DEFINITION
U+225E	□	MEASURED BY
U+225F	□	QUESTIONED EQUAL TO
U+2260	≠	NOT EQUAL TO
U+2261	≡	IDENTICAL TO
U+2262	≢	NOT IDENTICAL TO
U+2263	□	STRICTLY EQUIVALENT TO
U+2264	≤	LESS-THAN OR EQUAL TO
U+2265	≥	GREATER-THAN OR EQUAL TO
U+2266	≦	LESS-THAN OVER EQUAL TO
U+2267	≧	GREATER-THAN OVER EQUAL TO
U+2268	≨	LESS-THAN BUT NOT EQUAL TO
U+2269	≩	GREATER-THAN BUT NOT EQUAL TO
U+226A	≪	MUCH LESS-THAN
U+226B	≫	MUCH GREATER-THAN
U+226C	≬	BETWEEN
U+226D	≭	NOT EQUIVALENT TO
U+226E	≮	NOT LESS-THAN
U+226F	≯	NOT GREATER-THAN
U+2270	≰	NEITHER LESS-THAN NOR EQUAL TO
U+2271	≱	NEITHER GREATER-THAN NOR EQUAL TO
U+2272	≲	LESS-THAN OR EQUIVALENT TO
U+2273	≳	GREATER-THAN OR EQUIVALENT TO
U+2274	≴	NEITHER LESS-THAN NOR EQUIVALENT TO
U+2275	≵	NEITHER GREATER-THAN NOR EQUIVALENT TO
U+2276	≶	LESS-THAN OR GREATER-THAN
U+2277	≷	GREATER-THAN OR LESS-THAN
U+2278	□	NEITHER LESS-THAN NOR GREATER-THAN
U+2279	□	NEITHER GREATER-THAN NOR LESS-THAN

Punto di codifica	Aspetto	Descrizione
U+227A	≺	PRECEDES
U+227B	≻	SUCCEEDS
U+227C	≼	PRECEDES OR EQUAL TO
U+227D	≽	SUCCEEDS OR EQUAL TO
U+227E	≾	PRECEDES OR EQUIVALENT TO
U+227F	≿	SUCCEEDS OR EQUIVALENT TO
U+2280	⊀	DOES NOT PRECEDE
U+2281	⊁	DOES NOT SUCCEED
U+2282	⊂	SUBSET OF
U+2283	⊃	SUPERSET OF
U+2284	⊄	NOT A SUBSET OF
U+2285	⊅	NOT A SUPERSET OF
U+2286	⊆	SUBSET OF OR EQUAL TO
U+2287	⊇	SUPERSET OF OR EQUAL TO
U+2288	⊈	NEITHER A SUBSET OF NOR EQUAL TO
U+2289	⊉	NEITHER A SUPERSET OF NOR EQUAL TO
U+228A	⊊	SUBSET OF WITH NOT EQUAL TO
U+228B	⊋	SUPERSET OF WITH NOT EQUAL TO
U+228C	□	MULTISET
U+228D	□	MULTISET MULTIPLICATION
U+228E	⊎	MULTISET UNION
U+228F	⊏	SQUARE IMAGE OF
U+2290	⊐	SQUARE ORIGINAL OF
U+2291	⊑	SQUARE IMAGE OF OR EQUAL TO
U+2292	⊒	SQUARE ORIGINAL OF OR EQUAL TO
U+2293	⊓	SQUARE CAP
U+2294	⊔	SQUARE CUP
U+2295	⊕	CIRCLED PLUS
U+2296	⊖	CIRCLED MINUS
U+2297	⊗	CIRCLED TIMES
U+2298	⊘	CIRCLED DIVISION SLASH
U+2299	⊙	CIRCLED DOT OPERATOR
U+229A	⊚	CIRCLED RING OPERATOR
U+229B	⊛	CIRCLED ASTERISK OPERATOR
U+229C	□	CIRCLED EQUALS
U+229D	⊝	CIRCLED DASH
U+229E	⊞	SQUARED PLUS
U+229F	⊟	SQUARED MINUS
U+22A0	⊠	SQUARED TIMES
U+22A1	⊡	SQUARED DOT OPERATOR
U+22A2	⊢	RIGHT TACK
U+22A3	⊣	LEFT TACK
U+22A4	⊤	DOWN TACK
U+22A5	⊥	UP TACK
U+22A6	□	ASSERTION
U+22A7	□	MODELS
U+22A8	□	TRUE
U+22A9	⊩	FORCES
U+22AA	⊪	TRIPLE VERTICAL BAR RIGHT TURNSTILE
U+22AB	□	DOUBLE VERTICAL BAR DOUBLE RIGHT TURNSTILE
U+22AC	□	DOES NOT PROVE
U+22AD	□	NOT TRUE
U+22AE	⊮	DOES NOT FORCE
U+22AF	□	NEGATED DOUBLE VERTICAL BAR DOUBLE RIGHT TURNSTILE
U+22B0	□	PRECEDES UNDER RELATION
U+22B1	□	SUCCEEDS UNDER RELATION
U+22B2	◁	NORMAL SUBGROUP OF
U+22B3	▷	CONTAINS AS NORMAL SUBGROUP
U+22B4	⊴	NORMAL SUBGROUP OF OR EQUAL TO
U+22B5	⊵	CONTAINS AS NORMAL SUBGROUP OR EQUAL TO
U+22B6	□	ORIGINAL OF
U+22B7	□	IMAGE OF
U+22B8	⊸	MULTIMAP
U+22B9	□	HERMITIAN CONJUGATE MATRIX
U+22BA	⊺	INTERCALATE
U+22BB	⊻	XOR
U+22BC	⊼	NAND
U+22BD	□	NOR
U+22BE	□	RIGHT ANGLE WITH ARC
U+22BF	□	RIGHT TRIANGLE
U+22C0	⋀	N-ARY LOGICAL AND
U+22C1	⋁	N-ARY LOGICAL OR
U+22C2	⋂	N-ARY INTERSECTION
U+22C3	⋃	N-ARY UNION
U+22C4	⋄	DIAMOND OPERATOR
U+22C5	⋅	DOT OPERATOR
U+22C6	⋆	STAR OPERATOR
U+22C7	⋇	DIVISION TIMES
U+22C8	⋈	BOWTIE
U+22C9	⋉	LEFT NORMAL FACTOR SEMIDIRECT PRODUCT

Punto di codifica	Aspetto	Descrizione
U+22CA	⋊	RIGHT NORMAL FACTOR SEMIDIRECT PRODUCT
U+22CB	⋋	LEFT SEMIDIRECT PRODUCT
U+22CC	⋌	RIGHT SEMIDIRECT PRODUCT
U+22CD	⋍	REVERSED TILDE EQUALS
U+22CE	⋎	CURLY LOGICAL OR
U+22CF	⋏	CURLY LOGICAL AND
U+22D0	⋐	DOUBLE SUBSET
U+22D1	⋑	DOUBLE SUPERSET
U+22D2	⋒	DOUBLE INTERSECTION
U+22D3	⋓	DOUBLE UNION
U+22D4	⋔	PITCHFORK
U+22D5	□	EQUAL AND PARALLEL TO
U+22D6	⋖	LESS-THAN WITH DOT
U+22D7	⋗	GREATER-THAN WITH DOT
U+22D8	⋘	VERY MUCH LESS-THAN
U+22D9	⋙	VERY MUCH GREATER-THAN
U+22DA	⋚	LESS-THAN EQUAL TO OR GREATER-THAN
U+22DB	⋛	GREATER-THAN EQUAL TO OR LESS-THAN
U+22DC	□	EQUAL TO OR LESS-THAN
U+22DD	□	EQUAL TO OR GREATER-THAN
U+22DE	⋞	EQUAL TO OR PRECEDES
U+22DF	⋟	EQUAL TO OR SUCCEEDS
U+22E0	⋠	DOES NOT PRECEDE OR EQUAL
U+22E1	⋡	DOES NOT SUCCEED OR EQUAL
U+22E2	⋢	NOT SQUARE IMAGE OF OR EQUAL TO
U+22E3	⋣	NOT SQUARE ORIGINAL OF OR EQUAL TO
U+22E4	□	SQUARE IMAGE OF OR NOT EQUAL TO
U+22E5	□	SQUARE ORIGINAL OF OR NOT EQUAL TO
U+22E6	⋦	LESS-THAN BUT NOT EQUIVALENT TO
U+22E7	⋧	GREATER-THAN BUT NOT EQUIVALENT TO
U+22E8	⋨	PRECEDES BUT NOT EQUIVALENT TO
U+22E9	⋩	SUCCEEDS BUT NOT EQUIVALENT TO
U+22EA	⋪	NOT NORMAL SUBGROUP OF
U+22EB	⋫	DOES NOT CONTAIN AS NORMAL SUBGROUP
U+22EC	⋬	NOT NORMAL SUBGROUP OF OR EQUAL TO
U+22ED	⋭	DOES NOT CONTAIN AS NORMAL SUBGROUP OR EQUAL
U+22EE	⋮	VERTICAL ELLIPSIS
U+22EF	⋯	MIDLINE HORIZONTAL ELLIPSIS
U+22F0	□	UP RIGHT DIAGONAL ELLIPSIS
U+22F1	⋱	DOWN RIGHT DIAGONAL ELLIPSIS
U+22F2	□	ELEMENT OF WITH LONG HORIZONTAL STROKE
U+22F3	□	ELEMENT OF WITH VERTICAL BAR AT END OF HORIZONTAL STROKE
U+22F4	□	SMALL ELEMENT OF WITH VERTICAL BAR AT END OF HORIZONTAL STROKE
U+22F5	□	ELEMENT OF WITH DOT ABOVE
U+22F6	□	ELEMENT OF WITH OVERBAR
U+22F7	□	SMALL ELEMENT OF WITH OVERBAR
U+22F8	□	ELEMENT OF WITH UNDERBAR
U+22F9	□	ELEMENT OF WITH TWO HORIZONTAL STROKES
U+22FA	□	CONTAINS WITH LONG HORIZONTAL STROKE
U+22FB	□	CONTAINS WITH VERTICAL BAR AT END OF HORIZONTAL STROKE
U+22FC	□	SMALL CONTAINS WITH VERTICAL BAR AT END OF HORIZONTAL STROKE
U+22FD	□	CONTAINS WITH OVERBAR
U+22FE	□	SMALL CONTAINS WITH OVERBAR
U+22FF	□	Z NOTATION BAG MEMBERSHIP

Tabella u80.13. *Miscellaneous symbols.*

Punto di codifica	Aspetto	Descrizione
U+2600	□	BLACK SUN WITH RAYS
U+2601	□	CLOUD
U+2602	□	UMBRELLA
U+2603	□	SNOWMAN
U+2604	□	COMET
U+2605	★	BLACK STAR
U+2606	□	WHITE STAR
U+2607	□	LIGHTNING
U+2608	□	THUNDERSTORM
U+2609	□	SUN
U+260A	□	ASCENDING NODE
U+260B	□	DESCENDING NODE
U+260C	□	CONJUNCTION
U+260D	□	OPPOSITION
U+260E	☎	BLACK TELEPHONE
U+260F	□	WHITE TELEPHONE
U+2610	□	BALLOT BOX
U+2611	□	BALLOT BOX WITH CHECK
U+2612	□	BALLOT BOX WITH X
U+2613	□	SALTIRE

Punto di codifica	Aspetto	Descrizione
U+2614	□	UMBRELLA WITH RAIN DROPS
U+2615	□	HOT BEVERAGE
U+2616	□	WHITE SHOGI PIECE
U+2617	□	BLACK SHOGI PIECE
U+2619	□	REVERSED ROTATED FLORAL HEART BULLET
U+261A	□	BLACK LEFT POINTING INDEX
U+261B	☛	BLACK RIGHT POINTING INDEX
U+261C	□	WHITE LEFT POINTING INDEX
U+261D	□	WHITE UP POINTING INDEX
U+261E	☞	WHITE RIGHT POINTING INDEX
U+261F	□	WHITE DOWN POINTING INDEX
U+2620	□	SKULL AND CROSSBONES
U+2621	□	CAUTION SIGN
U+2622	□	RADIOACTIVE SIGN
U+2623	□	BIOHAZARD SIGN
U+2624	□	CADUCEUS
U+2625	□	ANKH
U+2626	□	ORTHODOX CROSS
U+2627	□	CHI RHO
U+2628	□	CROSS OF LORRAINE
U+2629	□	CROSS OF JERUSALEM
U+262A	□	STAR AND CRESCENT
U+262B	□	FARSI SYMBOL
U+262C	□	ADI SHAKTI
U+262D	□	HAMMER AND SICKLE
U+262E	□	PEACE SYMBOL
U+262F	□	YIN YANG
U+2630	□	TRIGRAM FOR HEAVEN
U+2631	□	TRIGRAM FOR LAKE
U+2632	□	TRIGRAM FOR FIRE
U+2633	□	TRIGRAM FOR THUNDER
U+2634	□	TRIGRAM FOR WIND
U+2635	□	TRIGRAM FOR WATER
U+2636	□	TRIGRAM FOR MOUNTAIN
U+2637	□	TRIGRAM FOR EARTH
U+2638	□	WHEEL OF DHARMA
U+2639	□	WHITE FROWNING FACE
U+263A	□	WHITE SMILING FACE
U+263B	□	BLACK SMILING FACE
U+263C	□	WHITE SUN WITH RAYS
U+263D	□	FIRST QUARTER MOON
U+263E	□	LAST QUARTER MOON
U+263F	□	MERCURY
U+2640	□	FEMALE SIGN
U+2641	□	EARTH
U+2642	□	MALE SIGN
U+2643	□	JUPITER
U+2644	□	SATURN
U+2645	□	URANUS
U+2646	□	NEPTUNE
U+2647	□	PLUTO
U+2648	□	ARIES
U+2649	□	TAURUS
U+264A	□	GEMINI
U+264B	□	CANCER
U+264C	□	LEO
U+264D	□	VIRGO
U+264E	□	LIBRA
U+264F	□	SCORPIUS
U+2650	□	SAGITTARIUS
U+2651	□	CAPRICORN
U+2652	□	AQUARIUS
U+2653	□	PISCES
U+2654	□	WHITE CHESS KING
U+2655	□	WHITE CHESS QUEEN
U+2656	□	WHITE CHESS ROOK
U+2657	□	WHITE CHESS BISHOP
U+2658	□	WHITE CHESS KNIGHT
U+2659	□	WHITE CHESS PAWN
U+265A	□	BLACK CHESS KING
U+265B	□	BLACK CHESS QUEEN
U+265C	□	BLACK CHESS ROOK
U+265D	□	BLACK CHESS BISHOP
U+265E	□	BLACK CHESS KNIGHT
U+265F	□	BLACK CHESS PAWN
U+2660	♠	BLACK SPADE SUIT
U+2661	♡	WHITE HEART SUIT
U+2662	♢	WHITE DIAMOND SUIT
U+2663	♣	BLACK CLUB SUIT
U+2664	□	WHITE SPADE SUIT
U+2665	♥	BLACK HEART SUIT
U+2666	♦	BLACK DIAMOND SUIT
U+2667	□	WHITE CLUB SUIT
U+2668	□	HOT SPRINGS
U+2669	□	QUARTER NOTE
U+266A	□	EIGHTH NOTE

Punto di codifica	Aspetto	Descrizione
U+266B	□	BEAMED EIGHTH NOTES
U+266C	□	BEAMED SIXTEENTH NOTES
U+266D	♭	MUSIC FLAT SIGN
U+266E	♮	MUSIC NATURAL SIGN
U+266F	♯	MUSIC SHARP SIGN
U+2670	□	WEST SYRIAC CROSS
U+2671	□	EAST SYRIAC CROSS
U+2672	□	UNIVERSAL RECYCLING SYMBOL
U+2673	□	RECYCLING SYMBOL FOR TYPE-1 PLASTICS (pete)
U+2674	□	RECYCLING SYMBOL FOR TYPE-2 PLASTICS (hdpe)
U+2675	□	RECYCLING SYMBOL FOR TYPE-3 PLASTICS (pvc)
U+2676	□	RECYCLING SYMBOL FOR TYPE-4 PLASTICS (ldpe)
U+2677	□	RECYCLING SYMBOL FOR TYPE-5 PLASTICS (pp)
U+2678	□	RECYCLING SYMBOL FOR TYPE-6 PLASTICS (ps)
U+2679	□	RECYCLING SYMBOL FOR TYPE-7 PLASTICS (other)
U+267A	□	RECYCLING SYMBOL FOR GENERIC MATERIALS
U+267B	□	BLACK UNIVERSAL RECYCLING SYMBOL
U+267C	□	RECYCLED PAPER SYMBOL
U+267D	□	PARTIALLY-RECYCLED PAPER SYMBOL
U+2680	□	DIE FACE-1
U+2681	□	DIE FACE-2
U+2682	□	DIE FACE-3
U+2683	□	DIE FACE-4
U+2684	□	DIE FACE-5
U+2685	□	DIE FACE-6
U+2686	□	WHITE CIRCLE WITH DOT RIGHT
U+2687	□	WHITE CIRCLE WITH TWO DOTS
U+2688	□	BLACK CIRCLE WITH WHITE DOT RIGHT
U+2689	□	BLACK CIRCLE WITH TWO WHITE DOTS
U+268A	□	MONOGRAM FOR YANG
U+268B	□	MONOGRAM FOR YIN
U+268C	□	DIGRAM FOR GREATER YANG
U+268D	□	DIGRAM FOR LESSER YIN
U+268E	□	DIGRAM FOR LESSER YANG
U+268F	□	DIGRAM FOR GREATER YIN
U+2690	□	WHITE FLAG
U+2691	□	BLACK FLAG
U+26A0	□	WARNING SIGN
U+26A1	□	HIGH VOLTAGE SIGN

Tabella u80.14. *Dingbats.*

Punto di codifica	Aspetto	Descrizione
U+2701	✁	UPPER BLADE SCISSORS
U+2702	✂	BLACK SCISSORS
U+2703	✃	LOWER BLADE SCISSORS
U+2704	✄	WHITE SCISSORS
U+2705		
U+2706	✆	TELEPHONE LOCATION SIGN
U+2707	✇	TAPE DRIVE
U+2708	✈	AIRPLANE
U+2709	✉	ENVELOPE
U+270A		
U+270B		
U+270C	✌	VICTORY HAND
U+270D	✍	WRITING HAND
U+270E	✎	LOWER RIGHT PENCIL
U+270F	✏	PENCIL
U+2710	✐	UPPER RIGHT PENCIL
U+2711	✑	WHITE NIB
U+2712	✒	BLACK NIB
U+2713	✓	CHECK MARK
U+2714	✔	HEAVY CHECK MARK
U+2715	✕	MULTIPLICATION X
U+2716	✖	HEAVY MULTIPLICATION X
U+2717	✗	BALLOT X
U+2718	✘	HEAVY BALLOT X
U+2719	✙	OUTLINED GREEK CROSS
U+271A	✚	HEAVY GREEK CROSS
U+271B	✛	OPEN CENTRE CROSS
U+271C	✜	HEAVY OPEN CENTRE CROSS
U+271D	✝	LATIN CROSS
U+271E	✞	SHADOWED WHITE LATIN CROSS
U+271F	✟	OUTLINED LATIN CROSS
U+2720	✠	MALTESE CROSS
U+2721	✡	STAR OF DAVID
U+2722	✢	FOUR TEARDROP-SPOKED ASTERISK
U+2723	✣	FOUR BALLOON-SPOKED ASTERISK
U+2724	✤	HEAVY FOUR BALLOON-SPOKED ASTERISK
U+2725	✥	FOUR CLUB-SPOKED ASTERISK
U+2726	✦	BLACK FOUR POINTED STAR
U+2727	✧	WHITE FOUR POINTED STAR
U+2728		
U+2729	☆	STRESS OUTLINED WHITE STAR
U+272A	✪	CIRCLED WHITE STAR

Punto di codifica	Aspetto	Descrizione
U+272B	✫	OPEN CENTRE BLACK STAR
U+272C	✬	BLACK CENTRE WHITE STAR
U+272D	✭	OUTLINED BLACK STAR
U+272E	✮	HEAVY OUTLINED BLACK STAR
U+272F	✯	PINWHEEL STAR
U+2730	✰	SHADOWED WHITE STAR
U+2731	✱	HEAVY ASTERISK
U+2732	✲	OPEN CENTRE ASTERISK
U+2733	✳	EIGHT SPOKED ASTERISK
U+2734	✴	EIGHT POINTED BLACK STAR
U+2735	✵	EIGHT POINTED PINWHEEL STAR
U+2736	✶	SIX POINTED BLACK STAR
U+2737	✷	EIGHT POINTED RECTILINEAR BLACK STAR
U+2738	✸	HEAVY EIGHT POINTED RECTILINEAR BLACK STAR
U+2739	✹	TWELVE POINTED BLACK STAR
U+273A	✺	SIXTEEN POINTED ASTERISK
U+273B	✻	TEARDROP-SPOKED ASTERISK
U+273C	✼	OPEN CENTRE TEARDROP-SPOKED ASTERISK
U+273D	✽	HEAVY TEARDROP-SPOKED ASTERISK
U+273E	✾	SIX PETALLED BLACK AND WHITE FLORETTE
U+273F	✿	BLACK FLORETTE
U+2740	❀	WHITE FLORETTE
U+2741	❁	EIGHT PETALLED OUTLINED BLACK FLORETTE
U+2742	❂	CIRCLED OPEN CENTRE EIGHT POINTED STAR
U+2743	❃	HEAVY TEARDROP-SPOKED PINWHEEL ASTERISK
U+2744	❄	SNOWFLAKE
U+2745	❅	TIGHT TRIFOLIATE SNOWFLAKE
U+2746	❆	HEAVY CHEVRON SNOWFLAKE
U+2747	❇	SPARKLE
U+2748	❈	HEAVY SPARKLE
U+2749	❉	BALLOON-SPOKED ASTERISK
U+274A	❊	EIGHT TEARDROP-SPOKED PROPELLER ASTERISK
U+274B	❋	HEAVY EIGHT TEARDROP-SPOKED PROPELLER ASTERISK
U+274C		
U+274D	❍	SHADOWED WHITE CIRCLE
U+274E		
U+274F	❏	LOWER RIGHT DROP-SHADOWED WHITE SQUARE
U+2750	❐	UPPER RIGHT DROP-SHADOWED WHITE SQUARE
U+2751	❑	LOWER RIGHT SHADOWED WHITE SQUARE
U+2752	❒	UPPER RIGHT SHADOWED WHITE SQUARE
U+2753		
U+2754		
U+2755		
U+2756	❖	BLACK DIAMOND MINUS WHITE X
U+2757		
U+2758	❘	LIGHT VERTICAL BAR
U+2759	❙	MEDIUM VERTICAL BAR
U+275A	❚	HEAVY VERTICAL BAR
U+275B	❛	HEAVY SINGLE TURNED COMMA QUOTATION MARK ORNAMENT
U+275C	❜	HEAVY SINGLE COMMA QUOTATION MARK ORNAMENT
U+275D	❝	HEAVY DOUBLE TURNED COMMA QUOTATION MARK ORNAMENT
U+275E	❞	HEAVY DOUBLE COMMA QUOTATION MARK ORNAMENT
U+2761	❡	CURVED STEM PARAGRAPH SIGN ORNAMENT
U+2762	❢	HEAVY EXCLAMATION MARK ORNAMENT
U+2763	❣	HEAVY HEART EXCLAMATION MARK ORNAMENT
U+2764	❤	HEAVY BLACK HEART
U+2765	❥	ROTATED HEAVY BLACK HEART BULLET
U+2766	❦	FLORAL HEART
U+2767	❧	ROTATED FLORAL HEART BULLET
U+2768	□	MEDIUM LEFT PARENTHESIS ORNAMENT
U+2769	□	MEDIUM RIGHT PARENTHESIS ORNAMENT
U+276A	□	MEDIUM FLATTENED LEFT PARENTHESIS ORNAMENT
U+276B	□	MEDIUM FLATTENED RIGHT PARENTHESIS ORNAMENT
U+276C	□	MEDIUM LEFT-POINTING ANGLE BRACKET ORNAMENT
U+276D	□	MEDIUM RIGHT-POINTING ANGLE BRACKET ORNAMENT
U+276E	□	HEAVY LEFT-POINTING ANGLE QUOTATION MARK ORNAMENT
U+276F	□	HEAVY RIGHT-POINTING ANGLE QUOTATION MARK ORNAMENT
U+2770	□	HEAVY LEFT-POINTING ANGLE BRACKET ORNAMENT
U+2771	□	HEAVY RIGHT-POINTING ANGLE BRACKET ORNAMENT
U+2772	□	LIGHT LEFT TORTOISE SHELL BRACKET ORNAMENT
U+2773	□	LIGHT RIGHT TORTOISE SHELL BRACKET ORNAMENT
U+2774	□	MEDIUM LEFT CURLY BRACKET ORNAMENT
U+2775	□	MEDIUM RIGHT CURLY BRACKET ORNAMENT
U+2776	❶	DINGBAT NEGATIVE CIRCLED DIGIT ONE
U+2777	❷	DINGBAT NEGATIVE CIRCLED DIGIT TWO
U+2778	❸	DINGBAT NEGATIVE CIRCLED DIGIT THREE
U+2779	❹	DINGBAT NEGATIVE CIRCLED DIGIT FOUR
U+277A	❺	DINGBAT NEGATIVE CIRCLED DIGIT FIVE
U+277B	❻	DINGBAT NEGATIVE CIRCLED DIGIT SIX
U+277C	❼	DINGBAT NEGATIVE CIRCLED DIGIT SEVEN
U+277D	❽	DINGBAT NEGATIVE CIRCLED DIGIT EIGHT
U+277E	❾	DINGBAT NEGATIVE CIRCLED DIGIT NINE
U+277F	❿	DINGBAT NEGATIVE CIRCLED NUMBER TEN

Punto di codifica	Aspetto	Descrizione
U+2780	①	DINGBAT CIRCLED SANS-SERIF DIGIT ONE
U+2781	②	DINGBAT CIRCLED SANS-SERIF DIGIT TWO
U+2782	③	DINGBAT CIRCLED SANS-SERIF DIGIT THREE
U+2783	④	DINGBAT CIRCLED SANS-SERIF DIGIT FOUR
U+2784	⑤	DINGBAT CIRCLED SANS-SERIF DIGIT FIVE
U+2785	⑥	DINGBAT CIRCLED SANS-SERIF DIGIT SIX
U+2786	⑦	DINGBAT CIRCLED SANS-SERIF DIGIT SEVEN
U+2787	⑧	DINGBAT CIRCLED SANS-SERIF DIGIT EIGHT
U+2788	⑨	DINGBAT CIRCLED SANS-SERIF DIGIT NINE
U+2789	⑩	DINGBAT CIRCLED SANS-SERIF NUMBER TEN
U+278A	❶	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT ONE
U+278B	❷	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT TWO
U+278C	❸	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT THREE
U+278D	❹	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT FOUR
U+278E	❺	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT FIVE
U+278F	❻	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT SIX
U+2790	❼	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT SEVEN
U+2791	❽	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT EIGHT
U+2792	❾	DINGBAT NEGATIVE CIRCLED SANS-SERIF DIGIT NINE
U+2793	❿	DINGBAT NEGATIVE CIRCLED SANS-SERIF NUMBER TEN
U+2794	➔	HEAVY WIDE-HEADED RIGHTWARDS ARROW
U+2795		
U+2796		
U+2797		
U+2798	➘	HEAVY SOUTH EAST ARROW
U+2799	➙	HEAVY RIGHTWARDS ARROW
U+279A	➚	HEAVY NORTH EAST ARROW
U+279B	➛	DRAFTING POINT RIGHTWARDS ARROW
U+279C	➜	HEAVY ROUND-TIPPED RIGHTWARDS ARROW
U+279D	➝	TRIANGLE-HEADED RIGHTWARDS ARROW
U+279E	➞	HEAVY TRIANGLE-HEADED RIGHTWARDS ARROW
U+279F	➟	DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
U+27A0	➠	HEAVY DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
U+27A1	➡	BLACK RIGHTWARDS ARROW
U+27A2	➢	THREE-D TOP-LIGHTED RIGHTWARDS ARROWHEAD
U+27A3	➣	THREE-D BOTTOM-LIGHTED RIGHTWARDS ARROWHEAD
U+27A4	➤	BLACK RIGHTWARDS ARROWHEAD
U+27A5	➥	HEAVY BLACK CURVED DOWNWARDS AND RIGHTWARDS ARROW
U+27A6	➦	HEAVY BLACK CURVED UPWARDS AND RIGHTWARDS ARROW
U+27A7	➧	SQUAT BLACK RIGHTWARDS ARROW
U+27A8	➨	HEAVY CONCAVE-POINTED BLACK RIGHTWARDS ARROW
U+27A9	➩	RIGHT-SHADED WHITE RIGHTWARDS ARROW
U+27AA	➪	LEFT-SHADED WHITE RIGHTWARDS ARROW
U+27AB	➫	BACK-TILTED SHADOWED WHITE RIGHTWARDS ARROW
U+27AC	➬	FRONT-TILTED SHADOWED WHITE RIGHTWARDS ARROW
U+27AD	➭	HEAVY LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
U+27AE	➮	HEAVY UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
U+27AF	➯	NOTCHED LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
U+27B1	➱	NOTCHED UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
U+27B2	➲	CIRCLED HEAVY WHITE RIGHTWARDS ARROW
U+27B3	➳	WHITE-FEATHERED RIGHTWARDS ARROW
U+27B4	➴	BLACK-FEATHERED SOUTH EAST ARROW
U+27B5	➵	BLACK-FEATHERED RIGHTWARDS ARROW
U+27B6	➶	BLACK-FEATHERED NORTH EAST ARROW
U+27B7	➷	HEAVY BLACK-FEATHERED SOUTH EAST ARROW
U+27B8	➸	HEAVY BLACK-FEATHERED RIGHTWARDS ARROW
U+27B9	➹	HEAVY BLACK-FEATHERED NORTH EAST ARROW
U+27BA	➺	TEARDROP-BARBED RIGHTWARDS ARROW
U+27BB	➻	HEAVY TEARDROP-SHANKED RIGHTWARDS ARROW
U+27BC	➼	WEDGE-TAILED RIGHTWARDS ARROW
U+27BD	➽	HEAVY WEDGE-TAILED RIGHTWARDS ARROW
U+27BE	➾	OPEN-OUTLINED RIGHTWARDS ARROW

Tabella u80.15. *Small form variants.*

Punto di codifica	Aspetto	Descrizione
U+FE50	﹐	SMALL COMMA
U+FE51	﹑	SMALL IDEOGRAPHIC COMMA
U+FE52	﹒	SMALL FULL STOP
U+FE54	﹔	SMALL SEMICOLON
U+FE55	﹕	SMALL COLON
U+FE56	﹖	SMALL QUESTION MARK
U+FE57	﹗	SMALL EXCLAMATION MARK
U+FE58	﹘	SMALL EM DASH
U+FE59	﹙	SMALL LEFT PARENTHESIS
U+FE5A	﹚	SMALL RIGHT PARENTHESIS
U+FE5B	﹛	SMALL LEFT CURLY BRACKET
U+FE5C	﹜	SMALL RIGHT CURLY BRACKET
U+FE5D	﹝	SMALL LEFT TORTOISE SHELL BRACKET
U+FE5E	﹞	SMALL RIGHT TORTOISE SHELL BRACKET
U+FE5F	﹟	SMALL NUMBER SIGN
U+FE60	﹠	SMALL AMPERSAND
U+FE61	﹡	SMALL ASTERISK
U+FE62	﹢	SMALL PLUS SIGN
U+FE63	﹣	SMALL HYPHEN-MINUS
U+FE64	﹤	SMALL LESS-THAN SIGN
U+FE65	﹥	SMALL GREATER-THAN SIGN
U+FE66	﹦	SMALL EQUALS SIGN
U+FE68	﹨	SMALL REVERSE SOLIDUS
U+FE69	﹩	SMALL DOLLAR SIGN
U+FE6A	﹪	SMALL PERCENT SIGN
U+FE6B	﹫	SMALL COMMERCIAL AT

Riferimenti

«

- *Unicode home page*
  http://www.unicode.org/
- *Unicode character database*
  http://www.unicode.org/Public/UNIDATA/

Stile di scrittura del sorgente

Blocchi di testo e rientri 537
Elenchi 537
Figure e tabelle 538
Titoli 539
Sezioni marcate 539

Il DTD di Alml suggerisce una logica nella stesura del sorgente. In questo capitolo si annotano dei suggerimenti sulla sistemazione degli elementi nel sorgente, allo scopo di ottenere una struttura ordinata, in funzione delle caratteristiche di questi.

Blocchi di testo e rientri

In generale, un blocco di testo viene scritto a partire dalla prima colonna del file, oppure viene incolonnato più a destra, di quattro caratteri alla volta, se si tratta di un sottoblocco di qualche tipo. Si osservi l'esempio seguente:

```
<frame>

    <p>Bisogna fare attenzione alle…
    …
    …</p>

</frame>
```

L'elemento '**frame**' serve a contenere uno o più blocchi interni; questi vanno indicati con un rientro.

Alla regola del rientro devono fare eccezione quei blocchi in cui lo spazio iniziale ha significato. In questo modo, gli elementi '**pre**', '**verbatimpre**', '**asciiart**' e '**syntax**' devono iniziare sempre dalla prima colonna.

I blocchi di testo con un contenuto di tipo lineare ed elementi interni a questo, dovrebbero mostrare la loro natura, avvolgendo il testo stesso, senza aggiungere rientri ulteriori. Per esempio, si usa l'elemento '**p**' in questo modo:

```
<p>Bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla…</p>
```

Al contrario, sarebbe spiacevole scrivere una cosa del genere:

```
<p>
    Bla bla bla bla…
</p>
```

I blocchi di testo, allineati in base alla necessità, vanno poi organizzati in modo da evitare di uscire dalla portata visiva di uno schermo normale; in pratica dovrebbero trovarsi entro le prime 80 colonne, come nell'esempio seguente:

```
<frame>

    <p>I blocchi di testo, allineati in base alla necessità, vanno poi
    organizzati in modo da evitare di uscire dalla portata visiva di uno
    schermo normale; in pratica dovrebbero trovarsi entro le prime
    <num>80</num> colonne.</p>

</frame>
```

Per favorire l'uso di funzionalità adatte del proprio programma di scrittura, allo scopo di reimpaginare i paragrafi e gli altri blocchi di testo, è necessario staccare i blocchi di testo tra di loro e dal loro contenitore, proprio come nell'esempio appena mostrato.

Elenchi

Gli elenchi di Alml sono definiti in modo da contenere sempre blocchi di testo. In tal modo, la struttura più coerente con quanto affermato a proposito dei rientri e dell'impaginazione dei blocchi, è quella dello schema seguente per ciò che riguarda gli elenchi puntati e numerati:

```
<ul>|<ol>
<li>

    blocco

    [blocco]
    …

</li>
[<li>

    blocco

    [blocco]
    …

</li>]
…
</ul>|</ol>
```

Per gli elenchi descrittivi, la situazione è abbastanza simile:

```
<dl>
<dt>voce</dt>
<dd>

    blocco

    [blocco]
    …

</dd>
[<dt>voce</dt>
<dd>

    blocco

    [blocco]
    …

</dd>]
…
</dl>
```

Figure e tabelle

Le figure interne al testo seguono la sorte di tutti gli altri elementi del genere, mentre le figure contenute nell'elemento '**object**' possono spostarsi sulla superficie della pagina. In questo senso, conviene indicarle sempre a partire dalla prima colonna, anche quando si chiede espressamente che rimangano fisse nella posizione in cui si trovano nel sorgente. L'elemento '**object**' è predisposto per contenere altri elementi, che però non è il caso di indicare con rientri. L'esempio seguente mostra la situazione comune in cui la figura è rappresentata dall'elemento '**img**'; in particolare merita attenzione la didascalia.

```
<object id="f-esempio-5">
<caption>

    Figura <objectref>. Bla bla bla…

</caption>
<imgblock>
<img imgfile="esempio-5" width="40%">
</imgblock>
</object>
```

In effetti, la didascalia è contenuta in un elemento '**caption**' che costituisce un blocco di testo. In precedenza è stata descritta la regola per cui i blocchi di testo devono essere realizzati ponendo il marcatore iniziale e quello finale in aderenza al testo contenuto, reimpaginando il tutto in base all'incolonnamento. Tuttavia, quello che si vede nell'esempio è lo stile proposto, che vale quindi come eccezione nel caso delle didascalie di figure, tabelle e listati.

Per le tabelle valgono le stesse considerazioni in relazione alle didascalie, mentre si propone una struttura particolare per l'elenco degli elementi che compongono le varie righe.


```
<object id="t-alml-isolat1-2">
<caption>

    Tabella <objectref>. Entità <special special="name">ISOlat1</special>:
    <bibref>added latin 1</bibref>. Seconda parte.

</caption>
<tabular col="3" columnfractions="0,309 0,191 0,500" border="1">
<thead>
  <trow>Macro SGML
<colsep>Risultato
<colsep>Descrizione
</trow>
</thead>
<tbody>
  <trow>&amp;ntilde;
<colsep>&ntilde;
<colsep>small n, tilde
</trow>
…
  <trow>&amp;yuml;
<colsep>&yuml;
<colsep>small y, dieresis or umlaut mark
</trow>
</tbody>
</tabular>
</object>
```

L'esempio mostra una situazione tipica. Si può osservare l'allineamento particolare del marcatore '**<trow>**' per avere il testo di tutte le celle della tabella allineato sulla stessa colonna del sorgente.

Allo scopo di facilitare la riorganizzazione di una tabella, è bene evitare di spezzare le righe di testo di una cella, quando queste superano la dimensione dello schermo.

Titoli

Gli elementi che contengono il titolo di una sezione (come per esempio '**tomeheading**', '**h0**', '**h1**', '**h2**', ecc.), vengono indicati nel sorgente secondo la struttura seguente, che mostra in particolare il caso del capitolo:

```
<h1 [id="stringa_identificativa"]>
titolo
[<indexentry [index="indice"]>voce_indice</indexentry>]
…
</h1>
```

Per facilitare un rielaborazione eventuale del sorgente, dovuta a una modifica del DTD di Alml, conviene lasciare il testo del titolo su una sola riga, anche se questo può essere lungo; inoltre, per lo stesso motivo, anche se il contenuto dell'elemento del titolo è di tipo lineare, conviene separare i marcatori dal testo del titolo, così come si vede dallo schema mostrato. Infine, per facilitare l'organizzazione delle voci da inserire nell'indice analitico, conviene collocare gli elementi '**indexentry**' preferibilmente nell'elemento del titolo, dopo il testo che lo descrive, in modo da guidare il lettore all'inizio della sezione che contiene la parola cercata.

Sezioni marcate

Le sezioni marcate devono essere delimitate correttamente e quando queste sono annidate, si possono creare problemi nel riconoscere la fine di questa o quella sezione. Per evitare ambiguità, è bene segnalare la macro dell'entità parametrica relativa:

```
<[%nome_entità_parametrica;[

    blocco_protetto

    [blocco_protetto]
    …

]]><!--%nome_entità_parametrica;-->
```

Quando una sezione marcata controlla una porzione di testo normale, è sufficiente che sia evidente l'ambito della sezione stessa. Per esempio:

```
<p>Bla bla bla <![%NS;[ciao ciao]]> bla bla bla...</p>
```


Estrapolazione di porzioni del file SGML 541
Esempio di un progetto 542
Aggregazioni 543

Di per sé, Alml nasce proprio per far fronte alle esigenze di un grande progetto di documentazione, pur essendo adatto anche a cose molto brevi. Il problema di un «grande progetto» non sta necessariamente nelle dimensioni di questo, quanto sulla gestibilità da parte di un singolo. È a questo proposito che Alml diventa veramente utile, in quanto consente di mettere tutte le proprie cose in un solo documento.

Solo mettendo assieme tutto, si ha la certezza di non perdere qualcosa. Forse non ci sarà la convenienza di pubblicare una raccolta che contiene ricette di cucina assieme a poesie e ad altri appunti, ma il singolo, ha sicuramente dei vantaggi a raccogliere tutto in un solo file SGML.

Si può obbiettare che il rischio di perdere i dati, se questi risiedono in un solo file, sia troppo alto. Ma se il problema è solo questo, basta avere l'accortezza di salvare usando un nome che contiene anche la data e un numero di serie (per esempio '`mio-20120131001.sgml`', '`mio-20120131002.sgml`', ecc.), controllando periodicamente le differenze tra il primo e l'ultimo file, prima di cancellare le copie obsolete ('**`diff -u mio-20120131001.sgml mio-20120131045.sgml | less`**'). Un'altra obiezione simile sta nella difficoltà di gestire un solo file enorme in un sistema CVS o simile, ma qui si parte dal presupposto che si tratti del lavoro di un singolo, il quale non ha alcuna convenienza a gestirselo tramite un sistema come quello.

Il vero problema, semmai, sta nel poter estrapolare delle porzioni del documento principale, per stampare o pubblicare solo ciò che serve (per esempio solo le ricette, solo le poesie, ecc.). In questo capitolo si vuole mostrare come si può organizzare il proprio lavoro in modo da mettere tutto assieme, con la possibilità di fare la composizione tipografica di una sola porzione che può servire per uno scopo preciso.

Estrapolazione di porzioni del file SGML

Alml include un programma realizzato in modo particolare per lo sviluppo di *a2*, con lo scopo di eseguire alcune operazioni di routine. Attraverso l'opzione '**`--sgml-extract`**' è possibile estrapolare una porzione di file SGML, delimitata con dei segni appropriati. Per esempio, si osservi il comando seguente:

```
$ a2engine --sgml-extract=sub-music.sgml example.sgml [Invio]
```

In questo modo, viene letto il file '`example.sgml`', collocato nella directory corrente, generando il file '`sub-music.sgml`', contenente le porzioni del file di partenza, delimitate tra i commenti speciali seguenti:

```
<!-- COPY TO "sub-music.sgml" START -->
…
…
<!-- COPY TO "sub-music.sgml" STOP -->
```

Naturalmente, le porzioni che generano un file, possono essere più di una, ripetendo le inserzioni appena mostrate.

Il file che contiene inserzioni di questo tipo, può indicare più blocchi con nomi diversi, che possono tranquillamente accavallarsi (a differenza delle sezioni marcate che possono solo annidarsi).

Una volta estratte le copie che servono del contenuto del file SGML principale, queste potrebbero essere aggregate assieme (anche attraverso comandi come '**`cat`**') in un altro file SGML temporaneo. In pratica, con qualche script si può organizzare il prelievo sistematico e la composizione tipografica di porzioni dedicate del lavoro

complessivo.

Esempio di un progetto

Nella documentazione che accompagna Alml c'è un esempio di un progetto di documentazione che prevede l'estrapolazione di porzioni più piccole: '`doc/example-project/`'. L'esempio è ridotto al minimo, ma serve a far comprendere il meccanismo.

Il documento complessivo è contenuto nel file '`example.sgml`' che ha la struttura seguente:

```
<!DOCTYPE ALML PUBLIC "-//D.G.//DTD Alml//EN"
[
<!ENTITY % NOTES "IGNORE">
]>
<alml lang="en" spacing="uniform">
<head>
    <admin>
        <description>An example for Alml documentation system</description>
        <keywords>SGML, XML, HTML, Alml</keywords>
    </admin>
    <title>Example to use Alml</title>
    <author>Pinco Pallino pinco.pallino@brot.dg</author>
    <date>2011.11.11</date>
    <legal>
        <p>Copyright &copy; Pinco Pallino, pinco.pallino@brot.dg</p>

        <p>Permission is granted to copy, distribute and/or modify this
        document under the terms of the GNU Free Documentation License,
        Version 1.1 or any later version published by the Free Software
        Foundation; with no Invariant Sections, with no Front-Cover
        Texts, and with no Back-Cover Texts. A copy of the license is
        included in the section entitled "GNU Free Documentation
        License".</p>
    </legal>
    <maincontents levels="2">Table of contents</maincontents>
</head>
<intro>
...
...
</intro>
<body>
...
...
</body>
<appendix>
...
...
</appendix>
<index>
<h1>
Index
</h1>

<printindex index="main">

</index>
</alml>
```

Inizialmente appare un'entità parametrica, da usare per isolare delle sezioni all'interno del documento, quindi inizia il contenuto del documento.

Si suppone di voler estrapolare due argomenti per poterne ottenere una composizione indipendente: vengono individuati i due blocchi per generare i file '`sub-music.sgml`' e '`sub-listings.sgml`'. Pertanto, nel sorgente principale vengono inseriti dei commenti di questo tipo:

```
<!-- COPY TO "sub-music.sgml" START -->
…
…
<!-- COPY TO "sub-music.sgml" STOP -->
```

```
<!-- COPY TO "sub-listings.sgml" START -->
…
…
<!-- COPY TO "sub-listings.sgml" STOP -->
```

Viene preparato un altro file, che inizia in modo simile a '`example.sgml`', ma che è privo di contenuti, in quanto è fatto per incorporare un file esterno, denominato '`sub-example-content.sgml`'. Inoltre, in questo file manca il titolo dell'opera, che viene letto da un file esterno, denominato '`TITLE`'. Si suppone che questo file che si affianca a '`example.sgml`', si chiami '`example-head.sgml`':

```
1  <!DOCTYPE ALML PUBLIC "-//D.G.//DTD Alml//EN"
2  [
3  <!ENTITY % NOTES "IGNORE">
4  <!ENTITY sub-example-content   SYSTEM "sub-example-content.sgml">
5  <!ENTITY WORKNAME              SYSTEM "TITLE">
6  ]>
7  <alml lang="en" spacing="uniform">
8  <head>
```



```
9      <admin>
10         <description>An example for Alml documentation system</description>
11         <keywords>SGML, XML, HTML, Alml</keywords>
12     </admin>
13     <title>&WORKNAME;</title>
14     <author>Pinco Pallino pinco.pallino@brot.dg</author>
15     <date>2011.11.11</date>
16     <legal>
17         <p>Copyright &copy; Pinco Pallino, pinco.pallino@brot.dg</email></p>
18
19         <p>Permission is granted to copy, distribute and/or modify this
20         document under the terms of the GNU Free Documentation License,
21         Version 1.1 or any later version published by the Free Software
22         Foundation; with no Invariant Sections, with no Front-Cover
23         Texts, and with no Back-Cover Texts. A copy of the license is
24         included in the section entitled "GNU Free Documentation
25         License".</p>
26     </legal>
27     <maincontents levels="2">Table of contents</maincontents>
28 </head>
29 <body>
30
31 &sub-example-content;
32
33 </body>
34 <index>
35 <h1>
36 Index
37 </h1>
38
39 <printindex index="main">
40
41 </index>
42 </alml>
```

Vanno osservate le righe 4 e 5, dove sono state aggiunte le dichiarazioni delle entità interne '**`sub-example-content`**' e '**`WORKNAME`**'. Nella riga 13 si vede l'utilizzo dell'entità '**`WORKNAME`**'; nella riga 31 si vede l'utilizzo di '**`sub-example-content`**'.

A questo punto si prepara una struttura di sottodirectory, per generare la composizione selettiva delle porzioni del documento principale. Si predispone precisamente '`projects/music/`' e '`project/listings/`'. In ognuna di queste due directory si predispongono dei collegamenti simbolici a tutto ciò che serve dalla directory principale, dove risiede il file SGML complessivo. Per esempio così:

```
example.sgml -> ../../example-head.sgml
Makefile -> ../../Makefile
pictures -> ../../pictures
```

Si osservi che in questo caso c'è un collegamento al file '`example-head.sgml`', che però è stato nominato convenientemente '`example.sgml`'. Infatti, si intende riutilizzare il file-make principale.

Nelle directory servono anche altri due file: '`TITLE`', che viene modificato in base al titolo che si vuole dare alla riduzione da comporre; inoltre serve uno script per attivare l'estrapolazione delle porzioni volute dal sorgente principale. Questo script potrebbe avere il contenuto seguente, che si riferisce precisamente all'estrazione di '`sub-music.sgml`':

```
#!/bin/sh
a2engine --sgml-extract=sub-music.sgml ../../example.sgml
mv -f sub-music.sgml sub-example-content.sgml
```

Come si comprende, è facile estrapolare anche porzioni più articolate dal sorgente principale, modificando in modo appropriato tale script; in pratica, alla fine occorre disporre di un solo file denominato '`sub-example-content.sgml`'.

Aggregazioni

Così come è possibile estrapolare qualcosa da un documento più complesso, è possibile anche aggregare in un documento già grande, delle porzioni di altri lavori (ammesso di averne ricevuto il permesso). Il meccanismo da usare è simile, in quanto si inseriscono dei commenti per l'estrapolazione delle porzioni desiderate nei file sorgenti esterni, quindi, nel sorgente che li deve accogliere, si dichiarano delle entità interne associate ai file di tali porzioni. Infine, si deve predisporre uno script appropriato, che, prima della composizione, esegua automaticamente l'estrazione di ciò che serve dagli altri documenti.

Questo meccanismo di aggregazione viene usato per *a2*, allo scopo di includere alcune opere di altri autori.

Questioni tecniche

Usare Textchk e Ispell con Alml 545
Espandere le potenzialità elaborative di TeX 545
Limiti strutturali di TeX 546
Soluzione attuata da Alml 547
Suddivisione automatica in volumi e parti della composizione finale PostScript e PDF 547
Programma di supporto 548

In questo capitolo vengono considerate alcune questioni che inizialmente non è necessario conoscere, ma che possono servire quando il proprio lavoro con Alml diventa significativo e ci si vuole organizzare di conseguenza.

Usare Textchk e Ispell con Alml

Textchk (sezione 47.10) può essere usati facilmente con Alml. In generale, si passa per una composizione in formato HTML singolo, quindi si utilizza questo programma. Supponendo di avere generato il file '`mio_file.html`':

```
$ textchk --input-type=html mio_file.html ↩
↪         mio_file.tchk mio_file.tdiag [Invio]
```

Per usare Ispell, è conveniente generare prima una versione del documento in formato testo puro. Per questo si potrebbe usare W3M, ma all'interno del pacchetto di Alml è disponibile un programma di supporto speciale, in grado di convertire opportunamente un file HTML per questo scopo. Si tratta di '**`alml-extra`**' che va usato con l'opzione '**`--html-to-text-for-spell`**':

```
alml-extra --html-to-text-for-spell < file_html > file_testo_non_formattato
```

In particolare, per evitare problemi con Ispell, nel file che si ottiene sono eliminate le barre oblique inverse ('\').

Naturalmente, usando poi Ispell nel file generato in questo modo, non ha senso fare delle correzioni, che invece vanno applicate manualmente al sorgente originale.

Espandere le potenzialità elaborative di TeX

Il file LaTeX generato da Alml tende a richiedere risorse impreviste a TeX. È molto probabile che per documenti di dimensioni medie, sia necessario espandere i limiti posti dalla configurazione di TeX.

In generale, si dovrebbe disporre di una distribuzione teTeX, per la quale si interviene nel file '`texmf/web2c/texmf.cnf`' (eventualmente potrebbe trattarsi meglio di '`/etc/texmf/texmf.cnf`', o simile).

Per la composizione di *a2* si è resa necessaria la modifica di alcune variabili; quello che si vede sotto sono i valori minimi da assegnare alle variabili rispettive:[1]

```
main_memory = 7000000
font_mem_size = 1000000
font_max = 2000
pool_size = 250000
hash_extra = 100000
buf_size = 100000
save_size = 40000
```

Si può tenere in considerazione l'abbinamento seguente, tra il rapporto generato da TeX e il file di configurazione '`texmf.cnf`', tenendo conto che in situazioni particolari il programma può segnalare la mancanza di una risorsa differente da quelle comuni:

```
Here is how much of TeX's memory you used:
42853 strings out of 55918
510063 string characters out of 647843
200381 words of memory out of 1000001
44744 multiletter control sequences out of 10000+40000
221835 words of font info for 188 fonts, out of 400000 for 1000
14 hyphenation exceptions out of 1000
```

Valore	Descrizione
`42853 strings out of 55918`	Dipende dalla variabile ***max_strings***. In questo caso gli è stato assegnato il valore 60000.
`510063 string characters out of 647843`	Dipende dalla variabile ***pool_size***. In questo caso gli è stato assegnato il valore 700000.
`200381 words of memory out of 1000001`	Dipende dalla variabile ***main_memory***. In questo caso gli è stato assegnato il valore 1000000.
`44744 multiletter control sequences ↩ ↪out of 10000+40000`	Il valore finale che si somma a 10000, dipende dalla variabile ***hash_extra***, a cui è stato assegnato il valore 40000.
`221835 words of font info for 188 fonts, ↩ ↪out of 400000 for 1000`	I due valori finali dipendono rispettivamente da '**`font_mem_size`**' e da '**`font_max`**'.
`14 hyphenation exceptions out of 1000`	Dipende dalla variabile ***hiph_size*** a cui corrisponde esattamente il valore finale.

Al termine delle modifiche a questo file, occorre ricordare di lanciare il comando '**`texconfig init`**', con i privilegi dell'utente '**`root`**':[2]

```
# texconfig init [Invio]
```

Nel caso particolare della distribuzione Debian, il file di configurazione '`/var/lib/texmf/texmf.cnf`' è ottenuto attraverso la fusione di file differenti, contenuti nella directory '`/etc/texmf/texmf.d/`'. In tal caso, per modificare le voci descritte in precedenza, occorre intervenire probabilmente nel file '`/etc/texmf/texmf.d/95NonPath.cnf`'; successivamente occorre eseguire il comando '**`update-texmf`**', il quale ricostruisce un file '`/var/lib/texmf/texmf.cnf`' nuovo; infine si deve eseguire '**`texconfig init`**'.

Si osservi comunque che nel pacchetto sorgente di Alml è disponibile il file '`etc/texmf/texmf.d/94alml.cnf`', che collocato correttamente nella directory '`/etc/texmf/texmf.d/`' risolve il problema senza intaccare gli altri file '`.cnf`' (richiedendo comunque l'avvio di '**`texconfig init`**', cosa che viene svolta automaticamente quando si installa il pacchetto Debian di Alml).

Limiti strutturali di TeX

Le distribuzioni normali di TeX potrebbero non essere in grado di gestire un gran numero di comandi '**`\label`**', anche se si tenta di intervenire nella configurazione. Questo si traduce in pratica in un limite insuperabile per ciò che nella configurazione viene mostrato come la variabile ***save_size***.

I comandi '**`\label`**' generano delle annotazioni in un file con estensione '`.aux`', simili all'esempio seguente:

```
\newlabel{anchor7}{{}{25}}
```

In questo caso si afferma che l'etichetta '**`anchor7`**' corrisponde alla pagina 25.

Generalmente, la composizione con i programmi '**`*tex`**' viene ripetuta per tre volte, allo scopo di acquisire le informazioni contenute in questo file: la prima volta viene costruito da zero, la seconda volta il testo viene reimpaginato utilizzando queste informazioni, rigenerandole nuovamente; infine, la terza volta non ci dovrebbero essere ulteriori spostamenti nell'impaginazione e il procedimento termina. Pertanto, la seconda e la terza volta viene letto il file con estensione '`.aux`'.

Sia i comandi '**`\label`**', sia i comandi '**`\newlabel`**' contenuti nel file ausiliario che viene incluso automaticamente, vanno a ridurre la memoria definita dalla variabile ***save_size***. Così succede normalmente che si riesce a completare la prima elaborazione del file, mentre nella successiva, caricando anche il file ausiliario la memoria non basta più. La segnalazione di errore tipica è la seguente:

```
! TeX capacity exceeded, sorry [save size=40000].
```

Di fatto, questa variabile non può superare il valore 65535, anche se si tenta di modificare i sorgenti di teTeX intervenendo nel file '`texk/web2c/tex.ch`'. Dovrebbe esserci una riga simile a quella seguente:

```
@!inf_save_size = 600;
@!sup_save_size = 40000;
```

Si può anche provare, aumentando il valore assegnato a '**`sup_save_size`**', per esempio come nel caso seguente, ma in pratica, il limite massimo che si riesce a raggiungere resta quello di 65535:[3]

```
@!inf_save_size = 600;
@!sup_save_size = 100000;
```

Soluzione attuata da Alml

Alml è un sistema di composizione pensato per la realizzazione di opere molto grandi, con indici generali e analitici gestiti autonomamente. In questo modo, la composizione tradizionale attraverso TeX genererebbe un file '`.aux`' con una quantità di voci molto grande. Per evitare di saturare il limite di TeX, questi riferimenti vengono inseriti in un altro file, con estensione '`.pageref`' e gestiti esternamente a TeX.

In breve, Alml gestisce le cose nel modo seguente.

1. Viene creato un file TeX in cui le etichette (le ancore) usano il comando '**`\AlmlLabel`**':

   ```
   \AlmlLabel{etichetta}
   ```

   Inoltre, i riferimenti alle pagine si fanno con comandi del tipo:

   ```
   \AlmlPageRef{0}{000}{etichetta}
   ```

2. Viene avviato TeX che elabora il file e genera un file '`.pageref`' in base ai comandi '**`\AlmlLabel`**'.
3. Viene letto il file '`.pageref`' e con quelle informazioni, il file TeX viene modificato intervenendo sui riferimenti alle pagine, che diventano:

   ```
   \AlmlPageRef{1}{pagina}{etichetta}
   ```

4. Si riavvia TeX che genera un nuovo file '`.pageref`'.
5. Viene letto il file '`.pageref`' e, con quelle informazioni, il file TeX viene modificato intervenendo sui riferimenti alle pagine, che diventano:

   ```
   \AlmlPageRef{2}{pagina}{etichetta}
   ```

6. Si riavvia TeX per l'ultima volta.

Suddivisione automatica in volumi e parti della composizione finale PostScript e PDF

Per facilitare la suddivisione della composizione PostScript in file contenenti solo un volume o solo una parte, vengono inserite nel sorgente TeX delle istruzioni per creare un file con estensione '`.pageloc`', contenente le informazioni necessarie:

```
BOF
tome{1}pageoffset{12}relativepage{1}
part{1}pageoffset{12}relativepage{7}
part{2}pageoffset{12}relativepage{19}
part{3}pageoffset{12}relativepage{105}
part{4}pageoffset{12}relativepage{121}
part{5}pageoffset{12}relativepage{171}
part{6}pageoffset{12}relativepage{203}
part{7}pageoffset{12}relativepage{269}
part{8}pageoffset{12}relativepage{319}
part{9}pageoffset{12}relativepage{351}
part{10}pageoffset{12}relativepage{383}
part{11}pageoffset{12}relativepage{411}
part{12}pageoffset{12}relativepage{415}
part{13}pageoffset{12}relativepage{469}
tome{2}pageoffset{12}relativepage{541}
part{14}pageoffset{12}relativepage{545}
eof{}pageoffset{12}relativepage{552}
EOF
```

Il significato dovrebbe essere intuitivo. Per esempio, il primo volume inizia dalla 13-esima pagina (ottenuta sommando 12 a 1) e termina all'inizio del volume successivo, ovvero alla 552-esima pagina (541+12−1). L'ultima pagina è la 564-esima.

In questo esempio, il valore 12 ricorrente rappresenta le pagine che precedono il contenuto vero e proprio del documento, in cui ci possono essere indici generali e introduzioni. Questo valore, definito qui come *page offset*, viene semplicemente sommato a quello finale.

Programma di supporto

Alml dispone di un programma di supporto, costituito dall'eseguibile '**`alml-extra`**', che consente di facilitare lo svolgimento di funzioni accessorie, in particolare per la riorganizzazione dei file PostScript.

```
alml-extra opzione [argomento]
```

A seconda dell'opzione utilizzata, può essere richiesto un argomento o meno, che fa riferimento a un file.

Dal momento che le opzioni che riguardano la conversione di file PostScript sono piuttosto difficili da ricordare, è disponibile anche uno script molto semplice che ne facilita l'uso:

```
alml-extra-menu file_ps
```

Figura u83.9. Aspetto di '**`alml-extra-menu`**', avviato con l'argomento '**`esempio.ps`**'.

```
.----------------------alml-extra OPTION esempio.ps-----------------------
|
| Select the option:
| .-------------------------------------------------------------------------
| |--a4-to-a5-2-a4              A4 to A5, no folding
| |--a4-to-a6-4-a4              A4 to A6, no folding
| |--a4-to-a5-2-a4-1h-1         A4 to A5, folded, signature 1
| |--a4-to-a5-2-a4-1h-10        A4 to A5, folded, signature 10
| |--a4-to-a6-4-a4-2h-2         A4 to A6, folded twice, signature 2
| |--a4-to-a6-4-a4-2h-4         A4 to A6, folded twice, signature 4
| |--a4-to-a6-4-a4-2h-6         A4 to A6, folded twice, signature 6
| |--a4-to-a6-4-a4-2h-8         A4 to A6, folded twice, signature 8
| |--a4-to-a6-4-a4-2h-10        A4 to A6, folded twice, signature 10
| |--a4-to-a6-4-a4-1v-1         A4 to A6, folded vertically, signature 1
| |--a5x4-to-a7x4               A5x4 to A7x4, no folding
| |--a5x4-to-a7x4-2-a4          A5x4 to A4, no folding
| |--a7x4-to-a7x4-2-a4          A7x4 to A4, no folding
| |--a7x4-to-a7x4-2-a4-1v-1     A7x4 to A7x4, folded vertically, signatu
| |--a7x4-to-a7x4-2-a4-1v-10    A7x4 to A7x4, folded vertically, signatu
| `-(+)-----------------------------------------------------------------
|
|-------------------------------------------------------------------------
|                      < OK   >            <Annulla>
`-------------------------------------------------------------------------
```

Le opzioni che vengono descritte nel seguito si riferiscono a '**`alml-extra`**', usato direttamente, dal momento che '**`alml-extra-menu`**' non prevede l'uso di opzioni proprie.

Opzione	Descrizione
`--help`	Mostra la guida rapida interna e conclude il funzionamento.
`--version`	Mostra le informazioni sulla versione e conclude il funzionamento.

Opzione	Descrizione
`--ps-group-pages=`***n_pagine***	Prevede che l'argomento finale sia un file PostScript, in cui vengono modificate le stringhe di definizione delle pagine, in modo che si possano individuare raggruppamenti di ***n*** pagine, di solito per facilitare la rilegatura. In pratica, in questo modo, si individuano più facilmente le pagine che compongono una segnatura.
`--ps-renumber-pages`	Prevede che l'argomento finale sia un file PostScript, in cui vengono modificate le stringhe di definizione delle pagine, in modo che la sequenza sia rinumerata a partire da uno.
`--alml-ps-split-tome=`***file_posizione_pagine***	Prevede che l'argomento finale sia un file PostScript, generato attraverso Alml, per il quale sia disponibile un file contenente la posizione di inizio dei vari volumi (dovrebbe trattarsi di un file con estensione '`.pageloc`'), che va indicato come argomento dell'opzione stessa. Quello che si ottiene sono diversi file PostScript, con estensione '`.`***n***`.ps`', dove in particolare '`.0.ps`' contiene le pagine precedenti al primo volume effettivo, con la presenza eventuale di file con estensione '`.app.ps`' e '`ndx.ps`', per le pagine delle appendici e degli indici analitici rispettivamente.
`--alml-dvi-split-tome=`***file_posizione_pagine***	Funziona come '**`--alml-ps-split-tome`**', ma si riferisce a file DVI.
`--alml-dvi-to-pdf-split-tome=`***file_posizione_pagine***	Funziona come '**`--alml-dvi-split-tome`**', ma converte il risultato in PDF.

Opzione	Descrizione
`--alml-ps-split-part=`***file_posizione_pagine***	Prevede che l'argomento finale sia un file PostScript, generato attraverso Alml, per il quale sia disponibile un file contenente la posizione di inizio dei vari volumi (dovrebbe trattarsi di un file con estensione '`.pageloc`'), che va indicato come argomento dell'opzione stessa. Quello che si ottiene sono diversi file PostScript, con estensione '`.`***n***`.ps`', dove in particolare '`.0.ps`' contiene le pagine precedenti alla prima parte effettiva, con la presenza eventuale di file con estensione '`.app.ps`' e '`ndx.ps`', per le pagine delle appendici e degli indici analitici rispettivamente.
`--alml-dvi-split-part=`***file_posizione_pagine***	Funziona come '**`--alml-ps-split-part`**', ma si riferisce a file DVI.
`--alml-dvi-to-pdf-split-part=`***file_posizione_pagine***	Funziona come '**`--alml-dvi-split-part`**', ma converte il risultato in PDF.
`--html-index=`***directory***	Genera, attraverso lo standard output, un file HTML che potrebbe essere utilizzato come file '`index.html`', contenente un elenco molto semplice dei file contenuti nella directory indicata.
`--html-index-basic=`***directory***	Come '**`--html-index`**', senza mostrare le date dei file.
`--html-index-basic-recursive`	Genera una serie di file '`index.html`', a partire dalla directory corrente e in tutte le sottodirectory.
`--html-to-text-for-spell`	Legge lo standard input, che dovrebbe essere costituito da un file HTML, filtrandolo allo scopo di generare un file di testo puro, utilizzabile per un controllo ortografico di qualche tipo. Il file che si ottiene viene emesso attraverso lo standard output.
`--perl-to-gettext`	Legge lo standard input, che dovrebbe essere costituito da un file sorgente Perl, filtrandolo allo scopo di generare un file di testo, adatto all'analisi da parte di Gettext, che solitamente riconosce bene solo le stringhe del linguaggio C. Il file che si ottiene viene emesso attraverso lo standard output.

Opzione	Descrizione
`--dos2unix`	Legge lo standard input, che dovrebbe essere un file di testo con interruzioni di riga in stile Dos (*<CR><LF>*), filtrandolo allo scopo di generare un file di testo con interruzioni di riga in stile Unix (*<LF>*). Il file che si ottiene viene emesso attraverso lo standard output.
`--unix2dos`	Legge lo standard input, che dovrebbe essere un file di testo con interruzioni di riga in stile Unix (*<LF>*), filtrandolo allo scopo di generare un file di testo con interruzioni di riga in stile Dos (*<CR><LF>*). Il file che si ottiene viene emesso attraverso lo standard output.
`--a4-to-a5-2-a4`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere due pagine A5 per ogni pagina A4 finale. Si ottiene un file con estensione '`.a5-2-a4.ps`'.
`--a4-to-a6-4-a4`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale. Si ottiene un file con estensione '`.a6-4-a4.ps`'.
`--a4-to-a5-2-a4-1h-1`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere due pagine A5 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, rilegando il tutto a segnature di un solo foglio. Si ottiene un file con estensione '`.a5-2-a4-1h-1.ps`'.
`--a4-to-a5-2-a4-1h-10`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere due pagine A5 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, rilegando il tutto a segnature di 10 fogli. Si ottiene un file con estensione '`.a5-2-a4-1h-10.ps`'.

Opzione	Descrizione
`--a4-to-a6-4-a4-2h-2`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, per due volte, rilegando il tutto a segnature di due fogli. In pratica, ogni segnatura si ottiene da un solo foglio A4 che viene piegato due volte. Si ottiene un file con estensione '`.a6-4-a4-2h-2.ps`'.
`--a4-to-a6-4-a4-2h-4`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, per due volte, rilegando il tutto a segnature di quattro fogli. In pratica, ogni segnatura si ottiene da due fogli A4 che vengono piegati assieme per due volte. Si ottiene un file con estensione '`.a6-4-a4-2h-4.ps`'.
`--a4-to-a6-4-a4-2h-6`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, per due volte, rilegando il tutto a segnature di sei fogli. In pratica, ogni segnatura si ottiene da tre fogli A4 che vengono piegati assieme per due volte. Si ottiene un file con estensione '`.a6-4-a4-2h-6.ps`'.
`--a4-to-a6-4-a4-2h-8`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, per due volte, rilegando il tutto a segnature di otto fogli. In pratica, ogni segnatura si ottiene da quattro fogli A4 che vengono piegati assieme per due volte. Si ottiene un file con estensione '`.a6-4-a4-2h-8.ps`'.

Opzione	Descrizione
`--a4-to-a6-4-a4-2h-10`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in orizzontale, per due volte, rilegando il tutto a segnature di 10 fogli. In pratica, ogni segnatura si ottiene da cinque fogli A4 che vengono piegati assieme per due volte. Si ottiene un file con estensione '`.a6-4-a4-2h-10.ps`'.
`--a4-to-a6-4-a4-1v-1`	Prevede che l'argomento finale sia un file PostScript, in formato A4, che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale, che deve essere piegata a metà, in verticale, rilegando il tutto a segnature di un foglio. Si ottiene un file con estensione '`.a6-4-a4-1v-1.ps`'.
`--a5x4-to-a7x4`	Prevede che l'argomento finale sia un file PostScript, in formato A5x4, che viene ridotto al formato A7x4. Si ottiene un file con estensione '`.a7x4.ps`'.
`--a5x4-to-a7x4-2-a4`	Prevede che l'argomento finale sia un file PostScript, in formato A5x4, che viene rielaborato in modo da ottenere due pagine A7x4 per ogni pagina A4 finale. Si ottiene un file con estensione '`.a7x4-2-a4.ps`'.
`--a7x4-to-a7x4-2-a4-1v-1`	Prevede che l'argomento finale sia un file PostScript, in formato A7x4, che viene rielaborato in modo da ottenere due pagine A7x4 per ogni pagina A4 finale, che deve essere piegata a metà, in verticale, rilegando il tutto a segnature di un foglio. Si ottiene un file con estensione '`.a7x4-2-a4-1v-1.ps`'.

Opzione	Descrizione
`--a7x4-to-a7x4-2-a4-1v-10`	Prevede che l'argomento finale sia un file PostScript, in formato A7x4, che viene rielaborato in modo da ottenere due pagine A7x4 per ogni pagina A4 finale, che deve essere piegata a metà, in verticale, rilegando il tutto a segnature di 10 fogli. Si ottiene un file con estensione '`.a7x4-2-a4-1v-10.ps`'.
`--a4s-to-a6s-4-a4s`	Prevede che l'argomento finale sia un file PostScript, in formato A4, orientato in modo orizzontale rovesciato (*seascape*) che viene rielaborato in modo da ottenere quattro pagine A6 per ogni pagina A4 finale. Si ottiene un file con estensione '`.a6s-4-a4s.ps`'.
`--a4s-to-a7s-8-a4`	Prevede che l'argomento finale sia un file PostScript, in formato A4, orientato in modo orizzontale rovesciato (*seascape*) che viene rielaborato in modo da ottenere otto pagine A6 per ogni pagina A4 finale. Si ottiene un file con estensione '`.a7s-8-a4.ps`'.

[1] La distribuzione GNU/Linux Debian organizza la configurazione del file '`texmf.cnf`' attraverso un insieme di file più piccoli, come viene descritto più avanti.

[2] Non tutte le modifiche che si apportano a questo file richiedono l'esecuzione di '**`texconfig init`**'; tuttavia è meglio ripeterlo, anche per quelle situazioni in cui non serve.

[3] Il limite strutturale sembra dipendere da un'organizzazione del programma pensata per l'elaborazione su architetture a 16 bit.

Gestione di «a2»

Articolazione dei file del sorgente 555
Inclusione selettiva dei file esterni ed entità speciali 555
Composizione guidata con il file-make 557

Questo capitolo descrive l'organizzazione del sorgente di *a2*, in modo da consentire una comprensione migliore del funzionamento di Alml.

Articolazione dei file del sorgente

Il sorgente di *a2* è composto da un file principale, molto grande, che fa riferimento ad altri file esterni per vari motivi:

```
.
|-- allegati/
|   '-- a2/
|
|-- riduzioni/
|   '-- ...
|
|-- figure/
|   |-- *.jpg
|   |-- *.pnm
|   |-- *.png
|   |-- *.tiff
|   '-- …
|
|-- video/
|   |-- *.ogv
|   |-- *.avi
|   '-- …
|
|-- ortografia/
|   |-- erroriecczezioni
|   |-- minimo.aff
|   |-- minimo.hash
|   |-- minimo.sml
|   |-- particolari
|   '-- vocabolario
|
|-- .textchk.rules      --> ortografia/erroriecczezioni
|-- .textchk.special    --> ortografia/particolari
|-- PAGINE
|-- EDIZIONE
|-- COPYING
|-- Makefile
|-- a2make
|-- a2sources
|-- a2sub.sgml
'-- a2-nnnnn.sgml
```

I file '`.textchk.rules`' e '`.textchk.special`', ovvero '`ortografia/erroriecczezioni`' e '`ortografia/particolari`', servono per l'uso di Textchk; mentre i file rimanenti nella directory '`ortografia/`' riguardano Ispell.

Inclusione selettiva dei file esterni ed entità speciali

L'inclusione dei file esterni, nel blocco principale, avviene per mezzo di istruzioni SGML del tipo seguente, dove si dichiara un'entità a cui si abbina il contenuto di un file intero:

```
<!ENTITY sub-samba-body                SYSTEM "sub-samba-body.sgml">
```

Altri pezzi ricorrenti di codice SGML sono dichiarati come entità interne, come questa:

```
<!ENTITY ALTRAILERTOMO.TEXT
'

    <unnumberedh1>
    Indice analitico del volume
    </unnumberedh1>

    <printindex index="main" indexcontext="tome">

'>
```

A seconda della circostanza, può essere necessario includere tali file o tali entità, oppure evitare la cosa. Per esempio, in una composizione che genera un file HTML unico non è il caso di ripetere certe informazioni sul copyright alla fine di ogni capitolo. Per questa e per altre ragioni, si utilizzano delle entità parametriche che nel sorgente vengono dichiarate in modo da disabilitarle:

```
<!ENTITY % HT  "IGNORE">
<!ENTITY % TT  "IGNORE">
<!ENTITY % PP  "IGNORE">
<!ENTITY % RM  "IGNORE">
<!ENTITY % NS  "IGNORE">
```

Queste entità parametriche controllano la dichiarazione di entità normali e l'inclusione di testo normale, come si può vedere nell'estratto semplificato che segue:

```
<![%PP;[
    <!ENTITY ALCOPYINGTOMO   "&ALCOPYINGTOMO.TEXT;">
    <!ENTITY ALCOPYINGPARTE  "&ALCOPYINGPARTE.TEXT;">
    <!ENTITY A2COPY          "&ALCOPY.TEXT;">
    <!ENTITY ALDEDICA        "&ALDEDICA.TEXT;">
]]>
```

Se tutte le entità parametriche viste in precedenza restano al valore originale ('**IGNORE**'), nessuna delle dichiarazioni che si vedono qui viene presa in considerazione. Se invece una di queste entità contiene il valore '**INCLUDE**', allora le dichiarazioni relative hanno significato.

Il sistema controlla l'abilitazione di queste entità parametriche attraverso l'opzione '**--sgml-include=*entità_parametrica***', come per esempio nel comando necessario a generare una composizione in PostScript:

```
$ alml --ps --verbose ↩
↪      --sgml-include=PP ↩
↪      --sgml-include=NS ↩
↪      mio_file.sgml [Invio]
```

Questa abilitazione preventiva prevale sulla dichiarazione di esclusione ('**IGNORE**') interna al sorgente e si ottiene il risultato desiderato.

Anche la dichiarazione delle entità normali segue la regola per cui vale ciò che è stato definito per primo. Pertanto, per evitare problemi, dopo la dichiarazione condizionata all'attivazione delle entità parametriche, viene ripetuta una dichiarazione di tali entità in modo predefinito:

```
<!ENTITY ALCOPYINGTOMO          "&ALCOPYINGTOMO.TEXT;">
<!ENTITY ALCOPYINGPARTE         "&ALCOPYINGPARTE.TEXT;">
<!ENTITY A2COPY                 "&ALCOPY.TEXT;">
<!ENTITY ALDEDICA               "&ALDEDICA.TEXT;">
```

Successivamente, nel corpo del file principale appare il richiamo alle entità relative per indicare il punto di inserzione del loro contenuto:

```
<tomeheading>
Primo approccio, architettura e filosofia del sistema operativo
</tomeheading>
&ALCOPYINGTOMO;

<h0>
Il software e le licenze
</h0>
&ALCOPYINGPARTE;

<h1>
...
```

Le tabelle u84.8 e u84.9 riepilogano le entità parametriche che controllano il sorgente di *a2* e le entità normali più importanti.

Tabella u84.8. Significato delle entità parametriche più importanti, usate nel sorgente di *a2*.

Macro SGML	Significato se attiva
`%HT;`	Composizione HTML normale.
`%HP;`	Composizione PDF normale.
`%TT;`	Composizione testo puro, su file unico.
`%PP;`	Composizione PostScript o PDF normale, ma per la stampa.
`%RM;`	Composizione con annotazioni per uso interno.
`%NS;`	Composizione completa di ciò che non viene controllato ortograficamente.

Tabella u84.9. Significato di alcune entità importanti, usate nel sorgente di *a2*.

Macro SGML	Contenuto
`&ALOPERA;`	Il nome dell'opera.
`&ALOPERAEMAIL;`	L'indirizzo o gli indirizzi di posta elettronica di riferimento.
`&ALPERIODO;`	L'anno o gli anni del copyright.
`&ALEDIZIONE;`	Edizione, scritta possibilmente come data.

Composizione guidata con il file-make

Il pacchetto dei sorgenti di *a2* include il file '`Makefile`', per facilitare la composizione dell'opera. La tabella u84.10 riepiloga i comandi principali.

Tabella u84.10. Comandi relativi al file-make di *a2*.

Comando	Risultato
`make clean`	Ripulisce da tutti i file non indispensabili.
`make check`	Analizza la sintassi SGML.
`make spell`	Utilizza Ispell per l'analisi del vocabolario.
`make textchk`	Utilizza Textchk per l'analisi sintattica.
`make urichk`	Utilizza Checkbot per il controllo degli URI.
`make draftdvi`	Composizione bozza in DVI.
`make dvi`	Composizione finale in DVI.
`make draftps`	Composizione bozza in PostScript.
`make ps`	Composizione finale in PostScript.
`make psl`	Composizione finale in PostScript A4 orizzontale, diviso in due colonne A5, con uno spazio aggiuntivo a sinistra per la rilegatura (stampa da un solo lato).
`make pstall`	Composizione finale in PostScript A5x4.
`make pstalla4`	Composizione finale in PostScript A4, diviso in due colonne A7x4.
`make draftpdf`	Composizione bozza in PDF.
`make pdf`	Composizione finale in PDF.
`make drafthtml`	Composizione bozza in HTML.
`make html`	Composizione finale in HTML.
`make html-text`	Composizione finale in HTML a pagina singola.
`make text`	Composizione finale in formato testo puro.

Si osservi che il formato ottenuto attraverso il comando '**make pstall**' va poi rielaborato con '**alml-extra**' (o '**alml-extra-menu**'), per riportarlo nell'ambito delle dimensioni di un foglio stampabile. In generale conviene convertirlo così:

```
$ alml-extra --a5x4-to-a7x4-2-a4 mio_file_ps [Invio]
```

Convenzioni di «a2»

Unità di misura e moltiplicatori 559
Casi particolari di testo che non viene enfatizzato 560
Valori numerici in lettere e in cifre 560
Distinzione nell'uso dei nomi degli applicativi 561
Descrizione degli acronimi 562
Indice analitico 562
Enfatizzazioni e uso degli elementi «special» 564
Rappresentazione del contenuto di file e dei flussi standard .. 568
Altri problemi di coerenza nell'uso degli elementi SGML 569
Sezioni marcate per le annotazioni 570

Questo capitolo raccoglie alcune convenzioni importanti relative all'opera *a2*. Le annotazioni sulla terminologia sono separate in un altro capitolo.

Unità di misura e moltiplicatori

In informatica si utilizzano delle unità di misura e dei moltiplicatori ben conosciuti, ma senza uno standard simbolico ben definito. Nel testo di questo documento si usano le convenzioni elencate nel seguito.

In particolare è bene distinguere tra il nome di un'unità di misura e il simbolo che la rappresenta: quando si parla dell'unità si usa il nome esteso, minuscolo; quando si indica un valore si deve usare il simbolo. In altri termini, si può parlare di hertz in generale, ma poi si indicano ***n*** Hz per indicarne una quantità precisa.

Quando si nominano i prefissi moltiplicatori, come «mega», «giga» e «tera», si usano le iniziali minuscole anche se il simbolo corrispondente è dato dalla loro iniziale maiuscola.

Unità di misura	Descrizione
byte, Kibyte, Mibyte, Gibyte, bit, Kibit, Mibit, Gibit	L'unità byte viene indicata al minuscolo, di seguito al suo moltiplicatore eventuale. In particolare: «Ki» sta per $2^{10} = 1024$; «Mi» sta per $2^{20} = 1048576$; «Gi» sta per $2^{30} = 1073741824$. L'unità di misura, con il suo moltiplicatore, viene indicata dopo e staccata dalla quantità a cui si riferisce.
bit/s, kbit/s, Mbit/s	L'unità bit/s (nota comunemente come bps, ovvero *Bit per second*) viene indicata al minuscolo, di seguito al suo moltiplicatore eventuale. In questo caso si utilizzano i moltiplicatori standard del SI: «k» sta per $10^3 = 1000$; «M» sta per $10^6 = 1000000$; «G» sta per $10^9 = 1000000000$. È importante ricordare che la lettera «k» deve essere minuscola. In generale, è preferibile la notazione bit/s rispetto a bps, perché la seconda è in realtà un'abbreviazione e come tale sconsigliabile secondo il SI. A questo proposito, si può leggere *Guide for the Use of the International System of Units (SI)* edito dal NIST (*National institute of standards and technology*), *http://physics.nist.gov/cuu/pdf/sp811.pdf*, in particolare la sezione 6.1.8: *Unacceptability of abbreviations for units*.

Unità di misura	Descrizione
Hz, kHz, MHz, GHz, THz	L'unità «hertz», il cui simbolo è «Hz», viene indicata nel modo che si vede, di seguito al suo moltiplicatore eventuale. In questo caso si utilizzano i moltiplicatori tradizionali: «k» sta per $10^3 = 1000$; «M» sta per $10^6 = 1000000$; «G» sta per $10^9 = 1000000000$; «T» sta per $10^{12} = 1000000000000$. È importante ricordare che la lettera «k» deve essere minuscola. Le unità di misura del SI, si nominano senza iniziale maiuscola. Tuttavia, il simbolo attribuito all'unità di misura è stato espresso con un'iniziale maiuscola quando questo derivava dal nome di una persona. Per esempio, questo è il caso di Hertz, di Alessandro Volta e di altri.
Ex	La grandezza Ex rappresenta l'altezza di una lettera «x», nell'ambito del sistema di composizione tipografica utilizzato. Viene indicata nel testo in questo modo, con l'iniziale maiuscola, per evitare confusione. Nel caso della misura relativa alla lettera «M» maiuscola, si usa il termine quadratone.

Casi particolari di testo che non viene enfatizzato

Alle volte verrebbe da enfatizzare di tutto. Qui si annotano le cose che per regola non vengono enfatizzate.

- **Valori numerici**

  I valori numerici di qualunque sistema di numerazione non vengono enfatizzati e i valori espressi in base diversa da 10 si indicano come si vede qui: $11 = 0B_{16} = 13_8 = 1011_2$. In particolare, le lettere alfabetiche utilizzate per le basi di numerazione superiori a 10, sono maiuscole.

- **Classi di indirizzi IPv4**

  Le classi di indirizzi IPv4 sono definite da lettere alfabetiche maiuscole che qui non vengono enfatizzate.

- **Indirizzi IPv4**

  Gli indirizzi numerici IPv4, a ottetti, vengono rappresentati così come sono, senza enfatizzazioni, utilizzando eventualmente il simbolo '`*`' per rappresentare l'indifferenza del valore di uno o più ottetti.

- **Indirizzi IPv6**

  Gli indirizzi numerici IPv6 vengono rappresentati così come sono, senza enfatizzazioni, utilizzando lettere minuscole.

- **Denominazione dei record di risorsa nel DNS**

  Le sigle usate nel DNS per identificare i record di risorsa dei file di definizione delle zone, sono scritti usando lettere maiuscole, senza enfatizzazioni.

- **Comandi del modem**

  I comandi AT e gli altri comandi dei modem vengono indicati utilizzando lettere maiuscole e senza enfatizzazioni. Ci possono essere eccezioni a questa regola, per esempio quando il contesto fa riferimento a una stringa che in quel caso particolare corrisponde proprio a un comando da inviare al modem.

Valori numerici in lettere e in cifre

I valori numerici da zero a nove vengono rappresentati preferibilmente in lettere, soprattutto per evitare ambiguità nella lettura, a meno che si presentino le condizioni seguenti:

- il numero è seguito da un simbolo (secondo il SI o anche altre convenzioni), per cui si preferisce lasciarlo espresso in cifre;

- il numero fa parte di un intervallo, dove l'altro valore è composto da due o più cifre, così si lascia in cifra anche il primo, dal momento che non ci possono essere ambiguità.

Distinzione nell'uso dei nomi degli applicativi

In generale, in questo documento, i nomi riferiti a degli «eseguibili», ovvero i programmi e gli script, sono indicati in modo evidenziato, esattamente come si utilizzano nel sistema operativo, senza cambiamenti nella collezione alfabetica delle lettere maiuscole e minuscole. Quando però il programma riveste un'importanza particolare, può assumere una denominazione diversa da quella che si usa nel nome del file eseguibile, oppure semplicemente si può decidere di trattarlo come qualcosa di più importante.

Per fare un esempio pratico, quando si parla di shell si fa riferimento alla shell Bash, alla shell Korn, alla shell C,... mentre l'eseguibile vero e proprio potrebbe essere '`bash`', '`ksh`', '`csh`',... Lo stesso vale per i programmi che meritano questa attenzione anche se il loro nome (verbale) non cambia.

In generale, il nome di un programma applicativo, di un pacchetto o di altre situazioni analoghe, viene indicato con l'iniziale maiuscola, salvo eccezioni che possono derivare dall'uso acquisito in una qualche forma differente, escludendo a ogni modo l'uso di sole lettere minuscole.

Il nome di un programma eseguibile va annotato in forma dattilografica, esattamente come deve essere scritto per avviarlo, ovvero come indicato nel file system. Nell'ambito dello stile dell'opera, quando si scrive il nome di un programma senza voler fare riferimento al file eseguibile, il nome in questione **non** può essere annotato usando solo lettere minuscole, anche se l'autore originale fa così.

La tabella u85.2 elenca alcune delle scelte di stile nell'uso dei nomi dei programmi distinguendo tra «eseguibile» e qualcosa di diverso: applicativo, pacchetto, servizio, sistema e simili, riferite a forme che costituiscono un'eccezione rispetto alla regola generale.

Tabella u85.2. Stile nell'uso dei nomi dei programmi distinguendo tra «eseguibile» e «applicativo», limitatamente ad alcune eccezioni.

Eseguibile	Applicativo, pacchetto, servizio, sistema,...
'`lilo`', '`grub`', '`syslinux`'	LILO, GRUB, SYSLINUX
'`*getty`'	Getty
'`getty`', '`uugetty`'	Getty_ps
'`mgetty`'	Mgetty+Sendfax
'`bash`'	shell Bash
'`csh`'	shell C
'`ksh`'	shell Korn
'`sh`'	shell Bourne
'`init`'	Procedura di inizializzazione del sistema, Init
'`cron`' (demone)	Cron (sistema)
'`inetd`'	supervisore dei servizi di rete
'`tcpd`'	TCP wrapper
'`portmap`'	Portmapper
'`named`'	BIND (pacchetto)
'`telnet`'	Telnet (programma) TELNET (protocollo o servizio)
'`finger`'	Finger (servizio)
'`sendmail`'	Sendmail
'`mail`'	Mailx
'`ex`'	EX
'`vi`'	VI
'`joe`'	Joe
'`m4`'	M4
'`mc`'	Midnight Commander
'`nsgmls`'	SP
'`sgmlspl`'	SGMLSpm
'`gs`'	Ghostscript

Eseguibile	Applicativo, pacchetto, servizio, sistema,...
'**bmv**'	BMV
'**ghostview**'	Ghostview
'**gv**'	GV
'**xpaint**'	XPaint
'**ee**', '**eeyes**'	Electric Eyes
'**xfm**'	XFM
'**tcd**', '**gtcd**'	TCD

Descrizione degli acronimi

«

Gli acronimi non sono sempre ottenuti con le sole iniziali delle parole che compongono il nome di qualcosa; inoltre, non c'è alcuna necessità pratica nell'evidenziare la corrispondenza tra le lettere usate e la frase corrispondente. In questo senso, la descrizione degli acronimi che si fa con l'elemento '**dacronym**' ha un aspetto uniforme: l'iniziale maiuscola e il resto del testo in minuscolo, tranne nel caso in cui si tratti di termini che rappresentano dei nomi importanti o degli altri acronimi, oppure quando la lingua di origine impone l'uso della maiuscola. Seguono alcuni esempi:

Acronimo	Descrizione completa	Annotazioni
MTA	*Mail transfer agent*	
XML	*Extensible markup language*	
ORF	*Österreichischer Rundfunk*	Nella lingua tedesca i sostantivi hanno l'iniziale maiuscola.
MIME	*Multipurpose Internet mail extentions*	Il nome che contiene (Internet) si scrive comunemente con l'iniziale maiuscola.

Indice analitico

«

Il problema della costruzione di un indice analitico è già trattato nel capitolo sullo stile letterario in generale. All'interno dell'opera *a2* ci sono delle particolarità che è bene precisare.

In particolare, l'indice analitico realizzato con il sistema di composizione di *a2* consente l'uso di un carattere dattilografico attraverso l'uso dell'elemento '**code**' e delle forme di evidenziamento particolari per combinazioni di tasti (reali o virtuali) e per codici ASCII:

```
<indexentry>Perl: <code>print</code></indexentry>
```

```
<indexentry><code>/etc/profile</code></indexentry>
```

```
<indexentry><kbd>Ctrl c</kbd></indexentry>
<indexentry><kbd>Ctrl \</kbd></indexentry>
```

```
<indexentry><vkbd>Control c</vkbd></indexentry>
<indexentry><vkbd>Control \</vkbd></indexentry>
```

```
<indexentry><asciicode>^c</asciicode></indexentry>
<indexentry><asciicode>ETX</asciicode></indexentry>
<indexentry><asciicode>^\</asciicode></indexentry>
```

- I termini inseriti nell'indice analitico vanno scritti usando lettere minuscole, a meno che si tratti di nomi particolari che vanno sempre scritti in un modo prestabilito.
  - La descrizione di un acronimo, inserita per esteso, si scrive con le stesse regole usate per l'elemento '**dacronym**', per cui l'iniziale è maiuscola.
  - Il nome di un applicativo, di un pacchetto, di un servizio, di un sistema e simili, va scritto nello stesso modo usato nel testo normale, senza cambiare lo stato delle lettere maiuscole e minuscole.
  - Il nome di file e directory va scritto esattamente come appare nel sistema operativo, utilizzando un carattere dattilografico, tenendo conto che i file eseguibili vanno indicati senza percorso, mentre gli altri dovrebbero contenerlo.
  - Il nome delle variabili di ambiente va scritto esattamente come appare nel sistema operativo (generalmente si tratta di nomi scritti con lettere maiuscole), usando un carattere dattilografico, lasciando il dollaro come prefisso.

  - Quando si inserisce il nome di un applicativo che possiede un eseguibile con lo stesso nome, non si annota anche il nome dell'eseguibile. In pratica, se si inserisce la voce «Pippo» senza enfatizzazione, non si annota anche la voce «pippo», corrispondente all'eseguibile omonimo, in modo dattilografico; al massimo, si inserisce un'altra volta la stessa voce «Pippo». Infatti, chi cerca notizie sul programma Pippo, o sull'eseguibile '**pippo**', si troverebbe in difficoltà nello scegliere tra l'una e l'altra voce. Quando invece un applicativo si articola in programmi eseguibili differenti, è sensato annotare sia il nome dell'applicativo, sia i nomi degli eseguibili che vengono descritti in modo particolare.
  - Quando la voce «Pippo» è comunque una cosa diversa da «pippo», le due voci vanno annotate esattamente e separatamente. Per esempio, si può fare riferimento al protocollo FTP e poi al programma eseguibile '**ftp**'. Il lettore può sentirsi confuso dalla distinzione, ma in tal caso è necessaria.
- Si utilizza il singolare, salvo eccezioni dovute al fatto che il termine al singolare possa intendersi come una cosa differente da ciò che si vuole realmente.
- La prima parola dovrebbe essere un sostantivo, o comunque è necessario sostantivare l'inizio della voce da inserire nell'indice analitico.
- Non si inizia una voce dell'indice analitico con un verbo; nel caso si può sostantivare il verbo. Per esempio, al posto di «salvare i dati» si può inserire la voce «salvataggio dei dati».
- Il sistema di composizione non consente l'indicazione di sottoclassificazioni nell'indice analitico, per cui si usa la tecnica seguente:

  voce* : *sottoclassificazione

  Questo fatto implica che i due punti vadano usati solo per questo scopo nelle voci dell'indice analitico; inoltre, diventa inopportuno l'inserimento di una sottoclassificazione ulteriore.
  - Una sottoclassificazione non è sottoposta all'obbligo di essere formulata usando il singolare; tuttavia, in caso di conflitto, si deve preferire la forma al singolare.
  - Una sottoclassificazione inizia con un sostantivo, così come iniziano le voci normali. Per esempio, «salvataggio: recuperare i dati» va sostituito con «salvataggio: recupero dei dati».
  - Non si usa il trattino per indicare una sottoclassificazione. Per esempio, «salvataggio -- recupero dei dati» va sostituto con «salvataggio: recupero dei dati».
- Quando si inserisce una voce in una sezione, non si inserisce nuovamente nelle sottosezioni relative. In pratica, se si inserisce la voce «Pippo» in corrispondenza dell'inizio di un capitolo, non si inserisce nuovamente la stessa voce in altre sezioni inferiori dello stesso capitolo.
- Le voci dell'indice analitico vanno inserite in riferimento alle sezioni opportune. Per esempio, la parola «file» potrebbe trovarsi in quasi tutte le pagine di un testo di informatica, mentre dovrebbe essere fatto un richiamo solo a quelle sezioni in cui si spiega di cosa si tratta (ammesso che ci sia).

I riferimenti per la generazione dell'indice analitico vanno posti preferibilmente nel titolo della sezione a cui fanno riferimento, come nell'esempio seguente:

```
<H3>
Copie di sicurezza
<indexentry>salvataggio: copia di sicurezza</indexentry>
<indexentry>salvataggio: recupero dei dati</indexentry>
</H3>
```

Come si vede, viene indicato prima il titolo e subito dopo l'elenco dei riferimenti da inserire nell'indice, che riguardano la sezione.

Inserendo le voci dell'indice analitico nell'ambito del titolo di una sezione, si comprende che non abbia senso ripetere la stessa voce nelle sottosezioni relative.

Enfatizzazioni e uso degli elementi «special»

La gestione corretta delle «enfatizzazioni» è sempre un problema serio di coerenza, soprattutto se si considera il fatto che l'enfatizzazione non implica solo la composizione finale con un aspetto particolare, ma anche la classificazione dell'oggetto per qualche fine. In particolare, l'elemento '**special**' non genera alcuna enfatizzazione, ma serve a dare una classificazione al termine inserito, per qualche ragione. L'opera *a2* usa le convenzioni che vengono sintetizzate in questa sezione.

- `<samp>`***stringa***`</samp>`

  Si usa all'interno di un testo normale per delimitare delle stringhe che hanno un valore letterale e si riferiscono in qualche modo a un'informazione tecnica. In particolare, si indicano in questo modo:

  – i nomi degli eseguibili;
  – gli esempi di opzioni di una riga di comando;
  – i nomi delle variabili di ambiente (senza il dollaro iniziale);
  – i nomi di elementi SGML (compreso XML e altre applicazioni);
  – gli esempi di istruzioni, comandi e direttive di qualunque tipo;
  – tutte le informazioni tecniche letterali che non ricadono in situazioni differenti.

- `<code>`***nome***`</code>`

  Si tratta di una forma di enfatizzazione molto simile a quella dell'elemento '**samp**', riservata a situazioni particolari:

  – può essere usata per ottenere un carattere dattilografico nelle voci dell'indice analitico;
  – l'elemento '**code**' può essere usato come **unico** elemento contenuto all'interno di '**dt**', quando in condizioni normali questo sarebbe stato rappresentato con l'elemento '**samp**';
  – l'elemento '**code**' può essere usato come **unico** elemento contenuto all'interno di '**faqh3**' o '**qh3**', quando in condizioni normali questo sarebbe stato rappresentato con l'elemento '**samp**'.

- `<file>`***file***`</file>`

  Nel testo normale, i nomi di file e directory, con o senza percorsi, vanno inseriti nell'elemento '**file**'. In generale, il nome di un file o di una directory dovrebbe sempre contenere l'informazione del percorso, salvo che si tratti implicitamente della directory corrente, oppure che non si possa stabilire una posizione precisa.

  Si usa la convenzione delle shell derivate da quella di Bourne, per cui il simbolo '**~/**' rappresenta la directory personale dell'utente che sta usando il sistema, mentre '***~utente/***' rappresenta la directory personale dell'utente indicato.

  In un percorso del genere si può inserire l'elemento '**var**', per descrivere una parte variabile dello stesso; inoltre è ammesso l'uso di caratteri jolly elementari, ovvero asterisco e punto interrogativo, per fare riferimento a più file.

  I nomi delle directory terminano sempre con la barra finale: '**/**' o '****' a seconda del sistema operativo a cui si fa riferimento.

  Quando si vuole fare riferimento a un file contenente un documento che dovrebbe essere raggiungibile in ogni sistema che abbia installato un certo applicativo, si può usare eventualmente l'elemento '**uri**', indicando un URI di tipo '**file:**', allo scopo di consentire l'accesso ipertestuale al file stesso. Naturalmente, ciò ha senso se l'URI che si indica è valido; quindi non è il caso di indicare caratteri jolly in un indirizzo del genere.

- `<var>`***metavariabile***`</var>`

  L'elemento '**var**' serve a delimitare una metavariabile, ovvero qualcosa che **descrive** ciò che va sostituito al suo posto. Non si indicano con questo elemento altri tipi di variabili, come potrebbero essere le variabili di ambiente o quelle di un programma scritto con un certo linguaggio. In tal caso, si userebbe piuttosto l'elemento '**samp**'.

  L'elemento '**var**' va usato prevalentemente all'interno dell'elemento '**syntax**', nei modelli sintattici, ma può essere usato utilmente anche dentro un elemento '**samp**', quando una parte della stringa non è fissa, così come in un elemento '**file**', per lo stesso motivo.

  Eccezionalmente, si può indicare un comando con l'inserzione di un elemento '**var**' all'interno del testo da digitare, ovvero l'elemento '**type**'. Tuttavia, in condizioni normali, si preferisce fare questo in un elemento '**syntax**', se il contesto lo consente.

  È consentita l'inserzione dell'elemento '**var**' anche all'interno di un elemento '**pre**', quando non è opportuno l'uso di un elemento '**syntax**' al suo posto.

  Il nome di una metavariabile dovrebbe descrivere ciò che rappresenta, mentre non deve essere un esempio del contenuto.

  Per evitare confusione, il nome va scritto usando possibilmente lettere minuscole, dove le varie parti possono essere separate da un trattino basso, come nel caso di ***mia_variabile***. Naturalmente si possono usare anche i numeri, purché sia chiaro che servono solo a individuare la metavariabile, come nel caso di ***nome_1***, ***nome_2***,... ***nome_n***. È da escludere l'uso di altri segni, perché creerebbero confusione, dal momento che i nomi delle variabili non appaiono delimitati. Se possibile è meglio evitare l'uso dell'apostrofo.

  Se possibile, è meglio comporre il nome delle metavariabili usando termini normali (non abbreviati o fusi assieme), in modo da non doverli inserire inutilmente nel vocabolario del controllo ortografico.

- `<dfn>`***definizione***`</dfn>`

  L'elemento '**dfn**' serve a delimitare una definizione, ovvero un termine che viene introdotto in riferimento a un contesto particolare. Va usato solo quando viene introdotto e non ha altro scopo che quello di generare una forma di evidenziamento uniforme.

  Lo stesso termine può apparire in contesti differenti e con un significato diverso; pertanto, l'uso dell'elemento '**dfn**' vale in quanto riferito al contesto particolare a cui appartiene la parola evidenziata.

  In generale, è bene evitare la proliferazione di evidenziamenti del genere, che vanno limitati alle situazioni in cui si vuole cogliere l'attenzione del lettore.

- `<strdfn>`***definizione_straniera***`</strdfn>`

  L'elemento '**strdfn**' serve a delimitare un termine o una definizione in lingua straniera, che non si intende utilizzare nel testo come terminologia normale, ma solo per spiegare, eventualmente, a cosa si sta facendo riferimento.

- `<em>`***testo***`</em>`
  `<strong>`***testo***`</strong>`
  `<small>`***testo***`</small>`
  `<big>`***testo***`</small>`

Le forme di evidenziamento generico vanno usate con molta parsimonia, perché non esiste una regola generale per il loro utilizzo. In particolare, un carattere ingrandito ottenuto con l'elemento '**big**' è utile nella realizzazione di presentazioni (lucidi per lavagna luminosa).

- `<bibref>`***titolo***`</bibref>`

Si usa l'elemento '**bibref**', nel testo normale, per delimitare il titolo di un documento o di un'opera di qualunque tipo.

- `<dacronym>`***descrizione_acronimo***`</dacronym>`

Si usa l'elemento '**dacronym**', nel testo normale, per delimitare la descrizione di un acronimo.

- `<acronym>`***acronimo***`</acronym>`

Questo elemento dovrebbe servire per delimitare un acronimo, secondo la logica del sistema di composizione, ma attualmente gli acronimi non vengono delimitati in alcun modo.

- `<kbd>`***combinazione_tasti***`</kbd>`

L'elemento '**kbd**' viene usato per indicare tasti (della tastiera) o combinazioni di tasti da premere. I nomi dei tasti vanno indicati come previsto (tabella u86.4, nel capitolo u86) e le combinazioni si ottengono inserendo uno spazio non interrompibile ('** **') tra i vari nomi o tra i simboli corrispondenti.

Nelle tabelle, quando si elencano tasti e combinazioni di tasti, si può fare a meno di questa forma di enfatizzazione.

- `<vkbd>`***combinazione_virtuale***`</vkbd>`

L'elemento '**vkbd**' viene usato per indicare tasti o combinazioni di tasti in forma virtuale. La denominazione segue abbastanza quella usata per la configurazione della tastiera della console dei sistemi GNU/Linux. Per esempio si può scrivere *<Control_c>* (che di solito si ottiene in pratica con la combinazione reale [*Ctrl c*]) e *<Meta_c>* (che di solito si ottiene con la combinazione reale [*Alt c*]).

- `<kp>`***tastiera_numerica***`</kp>`

L'elemento '**kp**' viene usato per indicare tasti premuti sulla tastiera numerica, all'interno dell'elemento '**kbd**'. Per esempio, '**<kbd>Ctrl Alt <kp>+</kp></kbd>**', indica la richiesta di premere i tasti «control», «alt» e il tasto «+» della tastiera numerica: [*Ctrl Alt* ⊞].

- `<button>`***pulsante_grafico***`</button>`

L'elemento '**button**' viene usato per indicare il nome di pulsanti grafici, anche in presenza di terminali a caratteri, che si selezionano attraverso un cursore o un puntatore grafico. Non si usa questo elemento per indicare l'uso della tastiera normale.

- `<menuitem>`***voce_di_menù***`</menuitem>`

Si delimitano in questo modo le voci di un programma grafico o di uno per terminali a caratteri che abbia un comportamento simile a quelli grafici, che siano riconducibili a scelte di un menù di funzioni. Rientrano in questa situazione i menù a tendina, i nomi delle etichette dei lembi di una sistema di cartelle, oppure il nome di un tipo di selezione che non sia riconducibile a un pulsante.

Questo elemento può essere usato anche per evidenziare le voci che rappresentano un tipo di casella di selezione, oppure le etichette dei campi in cui deve essere inserito qualche tipo di informazione.

- `<asciicode>`***nome_ascii***`</asciicode>`

Si delimitano in questo modo i nomi di caratteri speciali ASCII, che secondo la tradizione sono rappresentati da abbreviazioni con lettere maiuscole, così come le sequenze tradizionali derivate dalla telescrivente. La tabella u85.10 elenca tutti i caratteri che possono essere rappresentati in questo modo, mostrando anche il risultato dell'utilizzo dell'elemento.

La sequenza di più caratteri del genere si ottiene semplicemente mettendo a contatto più elementi '**asciicode**', come per esempio nel caso di *<CR><LF>*.

- `<uristr>`***uri_non_ipertestuale***`</uristr>`

L'elemento '**uristr**' si affianca all'elemento '**uri**', con lo scopo di rappresentare degli indirizzi URI per i quali non si vuole realizzare un riferimento ipertestuale. Ciò si rende necessario quando si scrive un indirizzo di fantasia o un indirizzo che si vuole conservare pur non essendo più valido. Si usa questo elemento anche quando si tratta di nomi a dominio, senza l'indicazione di una risorsa precisa.

- `<special special="name">`***nome***`</special>`

Serviva a delimitare, senza evidenziare, un nome, ma è in corso di eliminazione.

- `<special special="ttid">`***termine***`</special>`

Serve a delimitare, senza evidenziare, un termine particolare, espresso in italiano, per il quale si vuole avere un controllo. In generale ciò serve a seguire delle definizioni che non sono comuni ed è bene mantenere coerenti, per non confondere il lettore. Un'altra ragione per questo utilizzo è quello di facilitare la ricerca di tali definizioni nel momento in cui si decidesse di sostituirle con altre. Ciò si rende necessario perché un termine può avere quel certo significato speciale solo in un contesto particolare; pertanto, solo in questi casi va delimitato così.

I termini delimitati in questo modo sono evidenziati nel capitolo u86 con l'aggiunta di un asterisco.

- `<special special="ttsc">`***termine***`</special>`

Serve a delimitare, senza evidenziare, un termine particolare, espresso in inglese (o in un'altra lingua straniera), che per qualche ragione non sia traducibile, ma che non sia ancora stato acquisito completamente nella lingua italiana. L'elenco di questi termini si trova nella tabella u86.2 (capitolo u86).

- `<indexentry>…<code>`***stringa***`</code>…</indexentry>`

  `<indexentry>…<kbd>`***stringa***`</kbd>…</indexentry>`

  `<indexentry>…<vkbd>`***stringa***`</vkbd>…</indexentry>`

  `<indexentry>…<kp>`***stringa***`</kp>…</indexentry>`

  `<indexentry>…<asciicode>`***stringa***`</asciicode>…</indexentry>`

Nell'ambito delle voci dell'indice analitico, si possono usare alcuni elementi che comportano una forma di evidenziazione particolare. Si tratta di '**code**' (che va usato per tutte le situazioni in cui, nel testo normale si userebbe sia '**samp**', sia '**code**'), '**asciicode**', '**kbd**', '**vkbd**' e '**kp**'.

Tabella u85.10. Elenco dei caratteri speciali che si possono inserire nell'elemento '**asciicode**'.

Binario	Esadecimale	Ottale	Decimale	Carattere	Sigla equivalente
00000000_2	00_{16}	000_8	000_{10}	<*NUL*>	
00000001_2	01_{16}	001_8	001_{10}	<*SOH*>	<^*a*>
00000010_2	02_{16}	002_8	002_{10}	<*STX*>	<^*b*>
00000011_2	03_{16}	003_8	003_{10}	<*ETX*>	<^*c*>
00000100_2	04_{16}	004_8	004_{10}	<*EOT*>	<^*d*>
00000101_2	05_{16}	005_8	005_{10}	<*ENQ*>	<^*e*>
00000110_2	06_{16}	006_8	006_{10}	<*ACK*>	<^*f*>
00000111_2	07_{16}	007_8	007_{10}	<*BEL*>	<^*g*>
00001000_2	08_{16}	010_8	008_{10}	<*BS*>	<^*h*>
00001001_2	09_{16}	011_8	009_{10}	<*HT*>	<^*i*>
00001010_2	$0A_{16}$	012_8	010_{10}	<*LF*>	<^*j*>
00001011_2	$0B_{16}$	013_8	011_{10}	<*VT*>	<^*k*>
00001100_2	$0C_{16}$	014_8	012_{10}	<*FF*>	<^*l*>
00001101_2	$0D_{16}$	015_8	013_{10}	<*CR*>	<^*m*>
00001110_2	$0E_{16}$	016_8	014_{10}	<*SO*>	<^*n*>
00001111_2	$0F_{16}$	017_8	015_{10}	<*SI*>	<^*o*>
00010000_2	10_{16}	020_8	016_{10}	<*DLE*>	<^*p*>
00010001_2	11_{16}	021_8	017_{10}	<*DC1*>	<^*q*>
00010010_2	12_{16}	022_8	018_{10}	<*DC2*>	<^*r*>
00010011_2	13_{16}	023_8	019_{10}	<*DC3*>	<^*s*>
00010100_2	14_{16}	024_8	020_{10}	<*DC4*>	<^*t*>
00010101_2	15_{16}	025_8	021_{10}	<*NAK*>	<^*u*>
00010110_2	16_{16}	026_8	022_{10}	<*SYN*>	<^*v*>
00010111_2	17_{16}	027_8	023_{10}	<*ETB*>	<^*w*>
00011000_2	18_{16}	030_8	024_{10}	<*CAN*>	<^*x*>
00011001_2	19_{16}	031_8	025_{10}	<*EM*>	<^*y*>
00011010_2	$1A_{16}$	032_8	026_{10}	<*SUB*>	<^*z*>
00011011_2	$1B_{16}$	033_8	027_{10}	<*ESC*>	<^[>
00011100_2	$1C_{16}$	034_8	028_{10}	<*FS*>	<^\>
00011101_2	$1D_{16}$	035_8	029_{10}	<*GS*>	<^]>
00011110_2	$1E_{16}$	036_8	030_{10}	<*RS*>	<^^>
00011111_2	$1F_{16}$	037_8	031_{10}	<*US*>	<^_>
00100000_2	20_{16}	040_8	032_{10}	<*SP*>	
01111111_2	$7F_{16}$	177_8	127_{10}	<*DEL*>	

Rappresentazione del contenuto di file e dei flussi standard

In generale, il contenuto di un file o quanto emesso da un programma attraverso standard output e standard error, viene rappresentato in un elemento per il testo preformattato. Tuttavia, si manifestano dei problemi estetici, dovuti alla suddivisione del testo in pagine e al riconoscimento del contesto.

Per controllare la possibilità o meno di spezzare il testo tra più pagine, si inserisce l'elemento che lo contiene in un riquadro (l'elemento '**object**') fisso, che, a seconda di ciò che si preferisce, possa essere spezzato o meno:

```
<object pos="fixed" split="0">
…
…
</object>
```

In questo caso, evidentemente, si tratta di un listato che non si può spezzare; la scelta se mantenere unito o consentire la divisione in più pagine dipende naturalmente dalla lunghezza del testo.

Per quanto riguarda l'uso di linee e bordi di separazione, all'inizio del sorgente sono dichiarate alcune macro per la definizione dello stile, in modo da consentire in un secondo momento di cambiare l'aspetto generale. Si distinguono i casi seguenti, dimostrati da esempi:

- listato riferito al contenuto di un file su disco (che può essere anche uno script);

```
<object sep="none">
<verbatimpre border="1">
…
…
</verbatimpre>
</object>
```

```
<object sep="none">
<pre border="1">
…
…
</pre>
</object>
```

- listato riferito a quanto emesso attraverso lo standard output o lo standard error;

```
<object sep="none">
<verbatimpre border="0">
…
…
</verbatimpre>
</object>
```

```
<object sep="none">
<pre border="0">
…
…
</pre>
</object>
```

- listato riferito a quanto appare sullo schermo a seguito dell'utilizzo di un programma interattivo;

```
<object sep="none" split="0">
<verbatimpre border="1">
…
…
</verbatimpre>
</object>
```

```
<object sep="none" split="0">
<pre border="1">
…
…
</pre>
</object>
```

Altri problemi di coerenza nell'uso degli elementi SGML

La coerenza in ciò che poi si traduce in forme di enfatizzazione del testo è la cosa più importante da definire e anche la più difficile da mantenere. Tuttavia, ci sono altre considerazioni da fare su elementi che potrebbero sembrare più ovvi.

- I titoli della serie '**tomeheading**', '**h0**', '**h1**', '**h2**', '**h3**', '**h4**', '**testh1**', '**slideh1**', '**sheeth1**', '**faqh2**' e '**qh2**', vanno scritti senza inserire enfatizzazioni di alcun genere. Tuttavia, si possono e si devono inserire gli elementi '**special**'. In caso di necessità, si può delimitare qualche termine particolare solo usando le parentesi angolari uncinate standard.

  Come si vede, a questa regola fanno eccezione '**faqh3**' e '**qh3**' che invece possono contenere le enfatizzazioni comuni di un testo normale.

- Le tabelle vanno realizzate nel modo più semplice possibile, cercando di evitare contorsioni, allo scopo di facilitare la lettura anche a un utente che si limiti a scorrere il documento in forma di testo puro e semplice. Solo eccezionalmente è utile la realizzazione di tabelle HTML, racchiuse nell'elemento '**html**', per rappresentare schemi particolari, come nel caso delle schede riepilogative.

- Quando una figura può essere realizzata facilmente utilizzando semplicemente caratteri ASCII, conviene evitare la grafica, per consentire la visualizzazione della stessa anche in forma di testo puro. Si ottiene facilmente una figura del genere con l'elemento '**asciiart**', oppure anche solo con '**verbatimpre**'.

- A seconda dei tipi di composizione si possono avere pagine che hanno altezze molto diverse. Quando si realizza una tabella o una figura, occorre verificare che la composizione A4 normale

avvenga correttamente; di conseguenza sono poi corrette anche le altre forme.

Sezioni marcate per le annotazioni

Vengono usate delle sezioni marcate per inserire delle annotazioni da ottenere solo nella stampa di bozze. Queste sezioni marcate fanno riferimento all'entità parametrica '**RM**'. Di solito si fanno queste annotazioni utilizzando delle note a piè pagina. Si distinguono due tipi di segnalazioni: un'informazione da ricordare e un problema non risolto, da sistemare in un secondo momento. Si osservino i due esempi seguenti:

```
<![%RM;[<footnote><strong>ATTENZIONE</strong>:
questa notizia proviene da una ricerca fatta...
così e così...</footnote>]]><!--%RM;-->
```

```
<![%RM;[ <footnote><strong>SISTEMARE</strong>:
manca da analizzare la questione relativa
alla...</footnote>]]><!--%RM;-->
```


Glossario stilistico di «a2»

Termini tecnici particolari 572

Annotazioni su alcuni termini tecnici ritenuti «intraducibili» 574

Glossario 575

Unità temporali 576
Comandi e processi elaborativi 576
Memoria centrale e virtuale 579
Hardware 579
Dispositivi 579
Codifica 580
Tastiera 580
File di testo 581
Archiviazione e pacchetti applicativi 581
Dati 581
Crittografia e firma digitale 583
Linguaggi di programmazione e compilatori 583
Memoria di massa 585
Utenza 587
Documentazione 588
Interfaccia grafica 588
Rete e comunicazioni 589
Tipografia 591
Unicode 592
SGML/XML 593
Grafica 593
Usenet 593
Localizzazione 594
Varie 594

Forme espressive particolari 595

Annotazioni varie 595

Nomi dei caratteri speciali 596

Nomi da usare in modo uniforme 596

Riferimenti 597

Indice del glossario stilistico 597

Quando si scrivono documenti a carattere tecnico in lingua italiana, è difficile essere comprensibili, coerenti e anche corretti secondo le regole della lingua. Inoltre non si può nemmeno contare sulla presenza di una qualche autorità in grado di dare risposte a dei quesiti sul modo giusto di definire o di esprimere qualcosa.

Nella sezione 47.3 sono raccolti dei punti di riferimento, tuttavia resta aperto il problema della terminologia da adoperare. Attualmente, esiste la lista *tp (ad) lists·linux·it* che si occupa di discutere i problemi legati alle traduzioni di documenti come HOWTO, pagine di manuale e messaggi dei programmi GNU. La traduzione è una cosa differente dallo scrivere qualcosa di nuovo in italiano, comunque, la sensibilità e le scelte di ognuno possono essere diverse.

In questo capitolo si raccolgono alcune annotazioni sulle forme stilistiche ed espressive usate o che potrebbero essere usate in futuro in questa opera (nel tempo sono cambiate molte cose in questo documento e dovrebbero cambiarne ancora molte altre).

Sono sempre graditi i commenti riferiti al contenuto di questo capitolo e a tutto il resto dell'opera.

Alla fine del capitolo appare un indice analitico delle voci che sono state trattate qui. Ciò per facilitarne la ricerca, dal momento che i termini in questione appaiono secondo un certo ordine «logico», che non è quello alfabetico.

Nelle annotazioni delle sezioni seguenti, appaiono alcune sigle che hanno un significato molto semplice:

- *m.* -- maschile;
- *f.* -- femminile;
- *s.* -- singolare;
- *inv.* -- invariato al plurale;
- *agg.* -- aggettivo.

Il capitolo è organizzato secondo la struttura seguente:

Termini tecnici particolari 572

Annotazioni su alcuni termini tecnici ritenuti «intraducibili» 574

Glossario 575

Unità temporali 576
Comandi e processi elaborativi 576
Memoria centrale e virtuale 579
Hardware 579
Dispositivi 579
Codifica 580
Tastiera 580
File di testo 581
Archiviazione e pacchetti applicativi 581
Dati 581
Crittografia e firma digitale 583
Linguaggi di programmazione e compilatori 583
Memoria di massa 585
Utenza 587
Documentazione 588
Interfaccia grafica 588
Rete e comunicazioni 589
Tipografia 591
Unicode 592
SGML/XML 593
Grafica 593
Usenet 593
Localizzazione 594
Varie 594

Forme espressive particolari 595

Annotazioni varie 595

Nomi dei caratteri speciali 596

Nomi da usare in modo uniforme 596

Riferimenti 597

Indice del glossario stilistico 597

Termini tecnici particolari

«

Sono considerati acquisiti in italiano i termini tecnici elencati nella tabella u86.1. In quanto tali, sono indicati nel testo dell'opera e nel sorgente stesso senza enfatizzazioni tipografiche.

Tabella u86.1. Elenco dei termini tecnici considerati acquisiti nel linguaggio.

Termine	Annotazioni
bit	s. m. inv.
byte	s. m. inv.
computer	s. m. inv. -- meglio «elaboratore»
console	s. f. inv.
directory	s. f. inv.
sottodirectory	s. f. inv.
file	s. m. inv.
hardware	s. m. inv.
input	s. m. inv.
mixer	s. m. inv.
modem	s. m. inv.
monitor	s. m. inv.
mouse	s. m. inv.
output	s. m. inv.
routine	s. f. inv.
subroutine	s. f. inv.
software	s. m. inv.
standard input	
standard output	s. m. inv.
standard error	
timer	s. m. inv.
zoom	s. m. inv.

Inoltre, i termini che ormai sembrano far parte del linguaggio tecnico italiano in modo irrimediabile, sono annotati nella tabella u86.2. Anche questi appaiono nel testo dell'opera senza enfatizzazioni tipografiche, ma nel sorgente sono delimitati in modo da poter essere riconoscibili, attraverso la forma:

```
<special special="ttsc">termine</special>
```

Tabella u86.2. Elenco dei termini tecnici apparentemente consolidati in italiano, oppure che risultano intraducibili per qualche motivo. Nella tabella si annotano anche i termini che sarebbero traducibili, ma che hanno qualche particolarità se usati invariati in italiano.

Termine	Annotazioni
anycast	agg. -- IPv6
applet	s. f. inv. -- «applicazioncina»
array	s. m. inv.
bridge	s. m. inv.
gateway	s. m. inv.
router	s. m. inv.
broadcast	agg.
bus	s. m. inv.
cast	s. m. inv.
crontab	s. m. inv. -- file di Cron
dot-clock	s. m. inv.
driver	s. m. inv. -- meglio «gestore»
escape	s. m. inv. / agg.
feed	s. m. inv. -- Usenet
file system	s. m. inv. -- meglio evitare «filesystem»
firewall	s. m. inv.
firmware	s. m. inv.
fuzzy	agg. -- logica
hash	s. m. inv. -- array associativi di Perl
inode	s. m. inv.
join	s. m. inv. -- basi di dati
joystick	s. m. inv.
kernel	s. m. inv.
led	s. m. inv. -- i diodi led
link	s. m. inv. -- compilazione
linker	s. m. inv. -- compilazione
link-local	agg. -- IPv6
magic number	s. m. inv.
memoria cache	s. f. inv.
multicast	agg.
node-local	agg. -- IPv6
news	s. f. inv.
nice	agg. -- valore nice
organization-local	agg.
password	s. f. inv. -- qui si preferisce parola d'ordine
ping	s. m. inv. -- «fare il ping»
pixel	s. m. inv.
proxy	s. m. inv. -- se il contesto non è specifico, meglio parafrasare
record	s. m. inv.
script	s. m. inv.
shell	s. f. inv.

Termine	Annotazioni
subshell	s. f. inv.
site-local	agg. -- IPv6
socket	s. m. inv.
stack	s. m. inv. -- quello di un processo, per salvare i registri
task	s. m. inv. -- se possibile, meglio parafrasare
unicast	agg. -- IPv6
utility	s. f. inv. -- meglio «programma di servizio» o al limite «programma di utilità»

Le regole per la definizione del genere maschile o femminile per un termine tecnico proveniente dalla lingua inglese, che viene usato così com'è in italiano, sono molto vaghe. Inoltre, i termini inglesi che vengono incorporati nell'italiano vanno usati generalmente al singolare, anche quando esprimono quantità multiple.

Annotazioni su alcuni termini tecnici ritenuti «intraducibili»

- array

  Il termine array rappresenta una struttura di dati particolare, mentre i termini «vettore» e «matrice» sono specifici della matematica (si veda anche *Array*, *http://en.wikipedia.org/wiki/Array*).

- bridge; router; gateway

  Queste parole servono a definire in modo preciso e standard il ruolo di uno di quei nodi di rete che permettono un attraversamento tra una sottorete e un'altra.

- directory

  Il termine directory è stato tradotto in passato in vari modi poco soddisfacenti. Il concetto più elegante che si possa abbinare alla directory è quello di «cartella», che però è conveniente solo in presenza di un sistema operativo prevalentemente grafico.

- feed (Usenet)

  È difficile trovare una traduzione accettabile per esprimere il feed degli articoli di Usenet. Eventualmente si potrebbe parlare di «propagazione» degli articoli, quando il contesto lo consente, dal momento che non è proprio la stessa cosa.

- inode

  Si tratta di un termine costruito appositamente, anche se dalla fusione di termini inglesi. In particolare è difficile stabilire con certezza il significato della lettera «i» iniziale, probabilmente sta per *index*; comunque la diffusione del termine inode è tale per cui non avrebbe senso scomporlo e trasformarlo altrimenti. Per questo non è utile tentare di tradurlo, tanto più che si tratta di un nome costruito ad arte per rappresentare la caratteristica fondamentale dei file system Unix.

- magic number

  Il magic number, come descritto da *magic(4)*, è una realtà presente da molto tempo. Il concetto si avvicina a quello dell'impronta virale utilizzata dai programmi anti-virus, cosa che potrebbe essere descritta come una stringa di riconoscimento. Tuttavia, qualunque traduzione ne cancellerebbe la storia.

- memoria cache

  Memoria cache si usa generalmente così in italiano e non si può tradurre come «memoria tampone» che invece si riferisce al concetto di *buffer*. È da notare che «cache» viene dal francese e rappresenta qualcosa di nascosto o comunque celato. La traduzione «memoria di transito» può servire eventualmente come spiegazione, dal momento che rende abbastanza il concetto.

- news (Usenet)

  Questo termine è intraducibile e si riferisce al servizio offerto dalla rete Usenet: quello di distribuire le news. In questo senso, piuttosto che parlare di «servizio Usenet», è meglio riferirsi a un «servizio di gestione delle news».

- ping

  Il ping è inteso come l'azione di inviare una richiesta di eco a un nodo di rete, utilizzando il protocollo ICMP. In pratica, si fa il ping attraverso il comando '`ping`'. Dal momento che si tratta di un abbinamento con il ping-pong, sarebbe inopportuna la traduzione, a meno di volere essere più chiari, nel qual caso si può parlare di «richiesta di eco».

- pixel

  Dipende dal contesto: se il momento è discorsivo, si può tradurre come «punto grafico», tanto più che la dimensione di un punto del genere non è stabilita, ma dipende dalle caratteristiche del mezzo di visualizzazione.

- proxy

  Il proxy sarebbe il «procuratore» o il «procacciatore» di qualcosa. In italiano è improponibile l'uso di questo genere di traduzioni per indicare il concetto riferito ai servizi di un demone in un sistema operativo.

  Tuttavia, alle volte questo termine è utilizzato in situazioni che non sono particolarmente specifiche; in questi casi si potrebbe parlare di «intermediazione» e di «intermediario».

- record

  Questo termine viene usato spesso nel documento per indicare delle «righe» di file strutturate in campi, che contengono un'informazione completa su qualcosa.

- script

  Lo script, inteso come un programma scritto in un file di testo che viene eseguito per opera di un interprete, è un termine che non ha un equivalente in italiano nell'uso corrente. Ma si tratta di una parola di origine latina e non ci sono difficoltà particolari nell'inserimento in una frase in italiano, considerando che anche la pronuncia non è difficile.

- stack

  Il termine stack viene usato spesso per fare riferimento precisamente a quella parte di memoria utilizzata per salvare i registri del microprocessore nell'immagine dell'eseguibile, mentre questo è in funzione. Per rendere chiaro il concetto, conviene parlare di «stack del processo»; negli altri casi dovrebbe essere meglio utilizzare l'espressione «pila».

- standard input, standard output, standard error

  Si tratta di termini praticamente già tradotti, dove eventualmente si dovrebbero solo invertire le parole (input standard, output standard, ecc.). Ma in tal caso il problema starebbe nella trasformazione di standard error, che in questo modo diventerebbe «errore standard». Una forma del genere potrebbe far pensare all'«errore che fanno tutti», perché è «standard». Forse si potrebbe risolvere aggiungendo un trattino, ma poi occorrerebbe farlo anche per gli altri. Pertanto, più che tradurre, si può solo spiegare il significato di questi termini, attraverso una parafrasi, quindi si possono considerare intraducibili e acquisiti generalmente nel linguaggio.

- task

  Probabilmente, l'uso del termine task è inevitabile, a meno di grosse arbitrarietà linguistiche. Tra le altre cose, task ha il vantaggio di essere breve e facile da pronunciare all'interno di un testo italiano.

Glossario

Nelle sezioni seguenti sono annotati alcuni termini tecnici, nella maggior parte dei casi si tratta di termini in lingua inglese a cui si affiancano le loro traduzioni o traslazioni possibili in italiano, assieme a qualche commento. Le sezioni servono a distinguere i contesti.

L'asterisco che appare a fianco di alcune definizioni, serve a indicare quelle più deboli, o che comunque sono delimitate nel sorgente all'interno di elementi del tipo:

```
<special special="ttid">termine</special>
```

In questo modo sono più facili da tenere sotto controllo quando si stampa una bozza, senza lasciare tracce nella composizione finale standard.

Unità temporali

«

Le definizioni legate al conteggio del tempo rappresentano un concetto molto importante, specialmente per gli astronomi. In questo settore si sono sviluppati una serie di acronimi in lingua inglese, che a volte vengono anche tradotti in italiano. In generale, non è opportuno utilizzare acronimi tradotti, che comunque esistono.

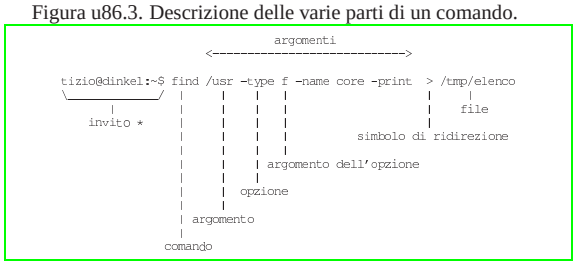
- UT, universal time ---> tempo universale
  È il tempo misurato con metodi astronomici, corrispondente al tempo solare medio del meridiano zero (quello passante per l'osservatorio astronomico di Greenwich)
- UTC, universal time coordinated ---> tempo universale coordinato
- CET ---> tempo medio dell'europa centrale
- CEST
  È l'ora estiva in anticipo di un'ora sul tempo CET.
- MET ---> CET
  MET è la vecchia sigla che è stata sostituita da CET.
- time zone ---> fuso orario
  zone ---> fuso
- daylight saving time ---> ora estiva
  È di uso comune chiamare «ora legale» l'orario anticipato di un'ora rispetto al tempo solare che si adotta dalla primavera all'autunno; tuttavia, sarebbe più corretto chiamarlo «ora estiva», chiamando corrispondentemente «ora invernale»[1] l'ora nel resto dell'anno, perché entrambe queste ore sono adottate per legge con tutti gli effetti civili, legali, ecc., quindi sono entrambe ore «legali». Perciò l'aggettivo «legale» non le differenzia.
- timestamp - -> informazione data-orario
  Il *timestamp* è il timbro contenente la data e l'ora dell'istante in cui questo timbro è stato fatto. La traduzione indicata rappresenta un modo imperfetto per esprimere il concetto. Il termine «datario» non è appropriato, dal momento che si riferisce allo strumento per timbrare e non al timbro che si ottiene; inoltre, serve a rappresentare una data, senza l'informazione oraria che invece è determinante nel termine inglese.
  Pare che nell'ambiente militare si usi la forma «gruppo data-orario».

Comandi e processi elaborativi

«

- riga di comando
  La riga di comando è quella riga che segue l'invito di una shell. La figura u86.3 raccoglie le definizioni riferite alle varie parti di questa riga.

Figura u86.3. Descrizione delle varie parti di un comando.



- prompt ---> invito
  In passato è stata usata la definizione «segnale di pronto» e anche «invito»; questa ultima forma ha il pregio di essere una buona traduzione del significato che ha *prompt*, anche se ha il difetto di non essere utilizzata in generale.
- utility ---> programma di utilità, programmi di utilità ---> utilità
  utility ---> programma di servizio
  In inglese si utilizza l'espressione «utility» per fare riferimento alla fornitura di servizi fondamentali come l'acqua, l'elettricità, il gas. In questo senso, dovrebbe essere più appropriata la traduzione programma di servizio, piuttosto di parlare di «utilità» come si è sempre fatto (non sapendo di cosa si tratta).
  Resta comunque necessario tenere presente che questa definizione non si può abbreviare semplicemente con «servizio», perché questo porterebbe a fare confusione con i servizi offerti da demoni, attraverso un socket di dominio Unix o una porta di rete.
- pipe, pipeline ---> condotto
  Si tratta dei condotti di programmi realizzati attraverso la shell.
- foreground (process) ---> (processo elaborativo) in primo piano
  Dal momento che l'uso in questa forma non è molto diffusa, anche se è abbastanza intuitiva, può essere opportuno indicare tra parentesi il termine originale in inglese almeno la prima volta.
- background (process) ---> (processo elaborativo) sullo sfondo
  Purtroppo, questa forma non è comprensibile immediatamente, per cui si può rendere necessario riproporre tra parentesi il termine originale in inglese almeno la prima volta, o comunque quando il contesto lo richiede per chiarezza.
- task
  Vedere i86.1.1.
- multitasking ---> multiprogrammazione ---> in multiprogrammazione ---> multiprogrammato
  Si tratta di un termine italiano di tipo accademico; probabilmente potrebbero andare bene forme del tipo «sistema che opera in multiprogrammazione» o semplicemente «sistema in multiprogrammazione», per tradurre il concetto di «sistema *multitasking*».
- singletasking --->monoprogrammazione ---> in monoprogrammazione ---> monoprogrammato
  Si riferisce a un sistema operativo che non funziona in multiprogrammazione.
- applicazione concorrente *
  Un programma che genera processi differenti gestiti simultaneamente (pseudo-simultaneamente).
  - applicazione multithread
    Un programma che si scinde in flussi di controllo (o flussi elaborativi) distinti, che però funzionano nello stesso contesto di dati. I flussi generati sono i ***thread*** a cui si fa riferimento.
  - applicazione parallela
    Un programma che si scinde in processi distinti, funzionanti in contesti indipendenti, comunicanti tra di loro attraverso dei messaggi.
  - applicazione distribuita
    Un programma che si scinde in processi distinti, eseguiti da macchine diverse, connesse in rete e comunicanti attraverso un protocollo appropriato.
- linguaggio concorrente *; linguaggio di programmazione concorrente *
  Il linguaggio di programmazione che consente la programmazione concorrente con appositi costrutti.
- programmazione concorrente *
  Programmazione di applicazioni concorrenti.

- multielaborazione *

  L'azione di un sistema composto da più CPU che lavorano assieme nello stesso elaboratore, oppure su elaboratori distinti connessi in rete.

- programma sequenziale

  Un programma che corrisponde a un processo singolo.

- runlevel ---> livello, livello di esecuzione
- exit status ---> valore di uscita
- boot ---> avvio, caricamento (del sistema operativo)
- Init ---> procedura di inizializzazione del sistema

  La definizione riguarda il sistema che controlla sia l'avvio che l'arresto del sistema.

  – procedura di avvio del sistema

    Questa forma viene usata per distinguere all'interno della procedura di inizializzazione del sistema la sequenza delle operazioni nel momento dell'avvio del sistema operativo.

  – procedura di arresto del sistema

    Questa forma viene usata per distinguere all'interno della procedura di inizializzazione del sistema la sequenza delle operazioni nel momento dell'arresto del sistema operativo.

- Init ---> processo iniziale

  Quando il contesto si riferisce al processo numero uno.

- shutdown ---> arresto del sistema
- spool ---> coda

  La traduzione non è perfetta, ma rappresenta il concetto.

- print job ---> processo di stampa
- shell job ---> gruppo di elaborazione
- log ---> registro, registro elettronico ---> registrazione degli eventi

  – to log ---> registrare

  – system log ---> registro del sistema

  – log file ---> file delle registrazioni *, file di registrazioni, file per le registrazioni

  – log archive ---> archivio delle registrazioni

  È da osservare che la forma «registro elettronico» viene usata frequentemente nei contratti e nei documenti formali.

- interrupt ---> interruzione

  In generale, la prima volta è meglio mettere tra parentesi il termine originale inglese.

- front-end - -> parte frontale *, - -> programma frontale

  back-end - -> parte terminale, - -> programma terminale

  La traduzione non è perfetta, dal momento che *front-end* e *back-end* rappresentano un concetto. In certe situazioni, il *back-end* può essere costituito da un gruppo di programmi, come nel caso delle copie di '**`postgres`**' avviate da '**`postmaster`**'. In questi casi, volendo continuare a parlare di programma terminale, occorrerebbe utilizzare il plurale.

  In certe situazioni, *front-end* viene usato in modo improprio anche in inglese; in quei casi, non ha senso la traduzione proposta qui.

- lock file ---> file lucchetto

  Un file lucchetto è un file che indica il blocco di un qualche tipo di risorsa (blocco perché la risorsa è impegnata in qualche modo e non è consentito l'accesso da parte di altri processi).

  Se c'è la possibilità di parafrasare, si potrebbe fare riferimento a un «file per il controllo dell'accesso», oppure a un «file di protezione» contro gli accessi concorrenziali a una risorsa data. Se poi non è necessario fare riferimento all'uso di questo file, ci si può riferire direttamente al fatto che si impedisce l'accesso da parte di altri processi, oppure che si protegge qualcosa contro gli accessi concorrenziali.

  Quando si parla di un blocco attraverso funzioni del sistema operativo, non è il caso di usare il termine *lock*, dal momento che «blocco» esprime perfettamente il concetto, anche per chi è esperto.

Memoria centrale e virtuale

«

- cache memory ---> memoria cache

  Vedere i86.1.1.

- buffer ---> memoria tampone

  La traduzione di *buffer* con «tampone» è interdisciplinare. Il termine *buffer*, tradotto con «tampone», si usa persino in chimica e biologia, rappresentando un concetto simile. Tuttavia, è meglio se quando si scrive si pensa che chi legge non sia necessariamente al corrente di questa ambivalenza, per cui conviene ricordare tra parentesi il termine inglese.

- swap ---> scambio

  Il contesto deve servire a comprendere il significato della parola «scambio». Per esempio: scambio della memoria, area di scambio (della memoria), partizione di scambio (della memoria) file di scambio (della memoria),...

- nvram ---> memoria non volatile

Hardware

«

- computer ---> elaboratore, sistema di elaborazione - -> sistema
- slot ---> alloggiamento

  Il termine *slot* può avere diverse traduzioni a seconda del contesto, pur restando nell'ambito dell'hardware. Per esempio, potrebbe essere espresso come «connettore» e anche «zoccolo», se si intende fare riferimento proprio al sistema di contatti e non anche allo spazio e alle guide delle schede che vi vengono inserite.

- controller ---> unità di controllo, scheda di controllo

  L'unità di controllo può essere una scheda o essere una parte integrata nella scheda madre. Al contrario, la scheda di controllo precisa che si tratta di una scheda distinta.

- terminale a caratteri, terminali a caratteri
- adapter, driver (inteso come unità hardware) ---> adattatore

  Questo è il caso di un'interfaccia hardware di qualche tipo, specialmente quando si tratta di una scheda. Si potrebbe parlare di «adattatore SCSI», «adattatore grafico»,...

  – scheda SCSI, interfaccia SCSI ---> adattatore SCSI

  – scheda video, scheda grafica ---> adattatore grafico

Dispositivi

«

In generale, si può distinguere tra dispositivo fisico e un dispositivo logico, per indicare rispettivamente l'hardware di un componente e il file di dispositivo relativo, che rappresenta la visione virtuale offerta dal kernel.

- device ---> dispositivo

  Distinguendo eventualmente in «fisico» o «logico», come accennato.

- device file ---> file di dispositivo
- device driver ---> gestore di dispositivo
- major number ---> numero primario
- minor number ---> numero secondario
- device number ---> numero di dispositivo

• driver ---> gestione di..., gestore *

In generale, se possibile è meglio parafrasare in modo da essere chiari sul significato della «gestione» a cui si fa riferimento. Si deve tenere presente che in alcune circostanze potrebbe non essere conveniente la traduzione.

• to drive ---> gestire

Codifica

«

• tab ---> carattere di tabulazione

• new-line ---> codice di interruzione di riga

Questa forma così prolissa serve a indicare il codice necessario a terminare una riga di un file di testo normale, in base alle esigenze del sistema operativo o comunque secondo il contesto. Ciò senza usare il termine *new-line*, che a volte alcuni autori di lingua inglese utilizzano per identificare precisamente il codice <*LF*>, indipendentemente da qualunque circostanza.

• escape

Non conviene tentare di tradurre il termine escape, soprattutto per la sua ambiguità, che lo fa utilizzare in tante situazioni. Vale la pena di annotare alcune forme tipiche in cui può essere utilizzato in italiano.

– codice di escape

Quando si tratta di una sequenza di escape che rappresenta qualcosa che esprime un codice speciale, come quello che non ha una corrispondenza simbolica (non è stampabile).

– sequenza di escape

Rappresenta qualcosa che si esprime con un carattere di «escape» iniziale, seguito da qualcosa d'altro. In generale, viene usata questa espressione in tutti i casi esclusi quelli in cui la sequenza di escape serve a rappresentare un codice particolare.

• eof, EOF ---> codice di EOF

EOF è un codice che di solito corrisponde a <*EOT*>, ma in generale dipende dalla piattaforma, più o meno come accade per il codice di interruzione di riga.

Tastiera

«

La tabella u86.4 raccoglie i nomi che sembrano più appropriati per i tasti delle tastiere comuni.

Tabella u86.4. Elenco dei nomi di alcuni tasti.

Originale inglese	Definizioni possibili in italiano
Esc, Escape	Esc
Return	Invio
Ctrl, Control	Ctrl, Controllo
Meta	Meta
Alt	Alt
Alt Gr	AltGr, Alt Gr
Shift	Maiuscole
Caps-lock	Fissa-maiuscole
Compose	Comp, Composizione
PgUp	Pagina su
PgDn	Pagina giù
Home	Inizio
End	Fine
Ins, Insert	Ins, Inserimento
Del, Delete	Canc, Cancellazione
Num Lock	BlocNum
Scroll Lock	BlocScorr
Print Screen	Stampa
Break	Interr, Interruzione
Pause	Pausa
F1, F2,...	F1, F2,... tasti funzione, tasti funzionali
Tab	Tab, Tabulazione -- per la dattilografia è «tabulatore»
Space	Barra spaziatrice, barra spazio, spazio

Le combinazioni di tasti vengono rappresentate usando il segno '**+**' per indicare una combinazione, mentre le sequenze di tasti vengono semplicemente elencate. Per esempio, [*Ctrl x*][*Ctrl y*] rappresenta la combinazione del tasto di controllo con la lettera «x», quindi il rilascio dei tasti e la combinazione successiva del tasto di controllo e della lettera «y». In presenza si combinazioni particolari, è bene spiegare tra parentesi ciò che si intende. Quando le combinazioni includono delle lettere alfabetiche, se non conta il fatto che siano maiuscole o minuscole, si rappresentano usando l'alfabeto minuscolo.

• key binding ---> associazione dei tasti *

Il significato attribuito a tasti particolari o a combinazioni di questi.

• interrupt character ---> carattere interrupt

Per comprenderne il senso, si può consultare la pagina di manuale *stty(1)*.

File di testo

«

• patch (file) ---> file di differenze

Trattando di *patch* si può parlare anche di «modifiche», «variazioni», «aggiornamenti» e simili, in base al contesto. Tuttavia, viene usata prevalentemente la definizione «file di differenze» come sostituto di «file di *patch*».

Quando si «applicano», si fa riferimento prevalentemente a «modifiche», senza richiamare nuovamente il termine «differenze».

• regular expression ---> espressione regolare

• '`/etc/motd`' ---> file contenente il messaggio del giorno

• '`/etc/issue`' --> file contenente il messaggio di pubblicazione

Sembra che il file '`/etc/issue`' servisse per fare apparire l'informazione sul nome e il numero di versione del sistema operativo. In questo senso, si potrebbe parlare di «numero di edizione», o di «pubblicazione», come se si trattasse di una rivista.

Archiviazione e pacchetti applicativi

«

• archive (file) ---> archivio ---> archivio compresso

Si fa riferimento a un file utilizzato per archiviare file e directory, come quello generato da '**`tar`**'. Un «archivio» è un file del genere realizzato in qualunque forma, anche compresso, mentre un «archivio compresso» è precisamente un file che ha subito una forma di riduzione (senza perdita).

Sono archivi anche i file dei pacchetti di applicazioni delle varie distribuzioni GNU/Linux: archivi Slackware, archivi RPM, archivi Debian...

– archiviazione

L'azione con cui si crea un archivio (compresso o meno che sia).

– estrazione (del contenuto)

L'azione con cui si estraggono i dati contenuti in un archivio (file, directory e altri oggetti, assieme ai loro attributi).

• package ---> pacchetto (applicativo)

In questo contesto, il «pacchetto» è ciò che è contenuto in un archivio di una distribuzione GNU/Linux. Per esempio, si può parlare di *archivio* '`bash_2.01.1-4.1.deb`' e di *pacchetto* '**`bash`**' (oppure Bash, se si vuole essere un po' meno precisi).

Dati

«

• magic number

Vedere i86.1.1.

• record

Vedere i86.1.1.

• standard input, standard output, standard error

Vedere i86.1.1.

• database ---> base di dati, basi di dati

In italiano si utilizza prevalentemente quando si tratta veramente di *database*, ovvero di *relazioni*. In italiano è frequente anche l'uso della forma «base dati», togliendo il «di».

– join ---> congiunzione *, giunzione *
– equijoin ---> equi-giunzione *
– outer-join ---> equi-giunzione incompleta * (a sinistra, a destra, totale)

• database ---> elenco, registro, tabella

Quando il termine *database* viene usato in modo improprio, potrebbe essere corretto l'uso di altri termini in funzione del contesto.

• data type ---> tipo di dati, tipi di dati

• checksum - -> codice di controllo

Il *checksum* indica letteralmente una «somma di controllo», solo che nel tempo si è esteso il suo significato includendo anche altre forme di controllo basate su operazioni di tipo diverso. A seconda delle circostanze si possono distinguere traduzioni differenti, che servono a precisare il tipo di controllo che viene attuato attraverso il *checksum*.

– codice di controllo
  Questa è probabilmente la traduzione migliore che potrebbe adattarsi alla maggior parte delle circostanze, dal momento che non viene specificato il modo in cui si ottiene il valore di controllo, non si stabilisce nemmeno la sua forma (numerica, alfabetica, ecc.); inoltre, non si stabilisce la sua dimensione.
– carattere di controllo, cifra di controllo *
  In tal caso il valore utilizzato per il controllo è rappresentato da un solo carattere, oppure precisamente da una cifra numerica.
– somma di controllo *
  Questa è la traduzione letterale del significato di *checksum*, però il suo uso dovrebbe essere riservato al caso in cui la funzione che genera il codice di controllo è basato su un procedimento di somme.
– campo di controllo *
  Quando l'informazione che funge da controllo è contenuta in un «campo».
– controllo
  Quando il contesto si riferisce all'azione di verificare qualcosa in base a un codice di controllo, ci si può limitare a usare il termine «controllo».

• MD5 digest, MD5 message digest - -> firma MD5

In un certo senso, un *MD5 digest* è un riassunto matematico di un messaggio, giustificando il motivo dell'utilizzo del termine *digest*. Oltre a questo, la stessa sigla «MD» sta per *Message digest*.

• upload, download ---> carico, scarico

I termini inglesi *upload* e *download* dovrebbero derivare dalle operazioni di carico e scarico delle merci dai mezzi di trasporto.

• octet ---> ottetto

• empty string ---> stringa nulla

• stringa vuota ---> stringa nulla

Per coerenza, è bene usare una sola definizione.

• trigger ---> grilletto

• overflow ---> traboccare

L'uso di «straripamento» è meno appropriato, date le dimensioni. Infatti, *overflow* si usa per le variabili, quando si creano dei riporti che non dovrebbero esserci, oppure per un testo che non rimane contenuto in un certo spazio (ma in tal caso potrebbe essere appropriato «debordare»).

• underflow ---> traboccare

La parola in questione è inventata ed è usata in contrapposizione a *overflow*; pertanto può avere valore solo in base al contesto. La traduzione come «traboccamento», va ovviamente associata a un aggettivo appropriato al contesto.

• bit rate ---> tasso del flusso di dati *

Il termine, spesso usato in inglese come se fosse una parola sola (*bitrate*), rappresenta un valore massimo o medio del flusso di dati di una sorgente sonora o video, compressa.

Crittografia e firma digitale

«

• in chiaro

cifrato, in cifra

Nel primo caso si fa riferimento a un'informazione che si presenta nella sua condizione normale, per la sua leggibilità o per l'accessibilità del suo contenuto; nel secondo caso, si tratta di un'informazione cifrata.

• cipher ---> cifratura

encrypted ---> cifrato

encryption ---> cifratura

La traduzione esatta di *encryption* è crittografia, che però è un sinonimo di cifratura. L'intenzione è quella di utilizzare in modo univoco questo tipo di tecnica.

• crittografia

Si preferisce riservare questo termine per fare riferimento al concetto generale, che si concretizza nell'uso della cifratura dei dati.

• decrittazione

Dovrebbe essere l'operazione attraverso cui si riesce a decifrare un'informazione senza conoscerne la chiave o il cifrario.

• Distinguishing Name, DN ---> nome distintivo *

Certificati X.509.

• Common Name, CN ---> nome comune *

Certificati X.509, campo CN del nome distintivo.

Linguaggi di programmazione e compilatori

«

I nomi attribuiti ai tipi di dati di ogni specifico linguaggio di programmazione, non possono essere tradotti, perché si tratta di parole chiave. Tuttavia, in un ambito discorsivo, ha senso utilizzare delle definizioni comprensibili. La tabella u86.5 mostra un elenco di quelle più comuni.

Tabella u86.5. Elenco delle definizioni possibili riferite ai tipi di dati più comuni.

char	carattere
int	intero
float	a virgola mobile (singola precisione)
double	a virgola mobile e doppia precisione

I nomi delle strutture di controllo del flusso e delle altre istruzioni che condizionano il flusso delle istruzioni, possono essere tradotti in alcuni casi, riferendosi al comportamento delle istruzioni a cui si fa riferimento. La tabella u86.6 riassume queste possibilità.

Tabella u86.6. Elenco delle definizioni e dei nomi riferiti alle strutture di controllo del flusso delle istruzioni.

go to	salto incondizionato
if	condizione, struttura condizionale
switch, case	selezione
while	iterazione, ciclo iterativo (condizione iniziale)
until	iterazione, ciclo iterativo (condizione finale)
for	iterazione enumerativa, ciclo enumerativo
break	salto, interruzione

La figura u86.7 raccoglie le definizioni riferite alla dichiarazione delle funzioni nei linguaggi di programmazione; la figura u86.8 fa riferimento alle definizioni utili nella chiamata di una funzione. Si osservi che il termine «parametro» non è equivalente ad «attributo», in quanto l'attributo è il valore che viene passato alla funzione, mentre il parametro è ciò che lo rappresenta formalmente (si veda anche *Parameter (computer science)*, *http://en.wikipedia.org/wiki/Parameter_(computer_science)*).

Figura u86.7. Linguaggi di programmazione: dichiarazione delle funzioni.

```
C               int potenza (int x, int y)
                (a)   (b)    (c)   (c)

Pascal          function potenza( x : integer; y : integer ) : integer;
                          (b)       (c)             (c)             (a)

Scheme          (define (potenza x y) ... )
                          (b)    (c)
(a) tipo restituito
(b) nome della funzione
(c) parametri formali
```

Figura u86.8. Linguaggi di programmazione: chiamata delle funzioni.

```
C               z = moltiplica (x, y);
                  (a)    (b)       (c)

Pascal          z := moltiplica( x, y );
                  (a)   (b)         (c)

Scheme          (set! z (moltiplica  x y))
                              (b)       (c)
(a) assegnamento
(b) funzione
(c) argomenti attuali (o parametri attuali): il contenuto delle variabili
    è ciò che costituisce gli argomenti attuali della chiamata
```

- assegnamento

  Per indicare il fatto che si assegna un valore a una variabile, si pone l'alternativa di usare «assegnazione» o «assegnamento». Si è scelta questa seconda alternativa.

- array

  Vedere i86.1.1.

- conversion specifier ---> specificatore di conversione

  Si tratta dei simboli che si utilizzano nelle funzioni quali ***printf()***, per descrivere il tipo di informazione che deve essere prelevata negli argomenti successivi e come deve essere formattata graficamente. Per esempio, nell'istruzione '`printf ("%d", 32);`', lo specificatore di conversione è la sequenza '`%d`'.

- associative array ---> array associativo

- parametro formale, parametro

  Nella dichiarazione di una funzione (o di una procedura), l'indicazione delle variabili di scambio, assieme alle informazioni sulle loro caratteristiche, viene indicata come la definizione dei ***parametri formali***.

  Quando si chiama una funzione, gli «argomenti» della chiamata, sono i ***parametri*** della funzione.

- preprocessor ---> precompilatore

  Quella parte del compilatore C che interpreta le direttive del tipo '`#include`' e simili, ovvero qualunque altro programma simile che ha un ruolo equivalente in altri linguaggi di programmazione.

- script

  Vedere i86.1.1.

- script language, scripting language ---> linguaggio script, linguaggio di script

- stream ---> flusso

  In questo caso, si fa riferimento allo *stream* che rappresenta un file aperto in C. Si distingue tra file aperto e file vero e proprio per il fatto che uno stesso file può essere stato aperto più volte all'interno di un programma.

- filehandle, file handle ---> flusso di file - -> flusso

  In questo caso, si fa riferimento a ciò che rappresenta un file aperto in Perl. Valgono le stesse considerazioni fatte per il caso dello *stream*, in C.

- makefile ---> file-make

  Questa definizione ha il vantaggio di essere comprensibile anche per chi utilizza abitualmente la definizione originale: *makefile*.

- to port ---> adattare

  porting ---> adattamento

  Con questo termine si fa riferimento al lavoro necessario per adattare un programma a un'altra piattaforma rispetto a quella di partenza.

- format ---> composizione

  Nel linguaggio C, le funzioni come ***printf()*** utilizzano una stringa, nota come *format string*, che può essere tradotta come «stringa di composizione», in quanto si tratta proprio di un procedimento di trasformazione in simboli tipografici. Lo stesso ragionamento vale per le funzioni come ***scanf()*** che partono da un'informazione in formato tipografico, per estrapolare i dati in essa contenuti.

Memoria di massa

«

- hard disk ---> disco fisso

  Il «disco fisso» è quel tipo di disco che fa parte integrante dell'unità che si occupa di accedere ai suoi dati e si distingue dal «disco rimovibile» che invece ne è indipendente. Il termine *hard disk* viene tradotto spesso come disco rigido, probabilmente in contrapposizione al dischetto che originariamente è stato realizzato su una superficie flessibile; tuttavia questa non sembra una buona ragione per usare il termine «disco rigido» perché esistono «dischetti» realizzati su superficie rigida, ma soprattutto perché i primi dischi rimovibili sono stati realizzati su superficie di alluminio.

  In base a queste considerazioni, anche un disco non rimovibile innestato su un'unità esterna, USB o SCSI, è da considerare a tutti gli effetti un disco fisso.

- format ---> formattazione ---> inizializzazione

  In generale, il verbo «inizializzare» è più appropriato, specificando eventualmente se si tratta di inizializzazione a basso livello (quando vengono collocate le tracce) o ad alto livello (quando viene predisposto il file system).

  Si preferisce usare il termine «composizione» in ambito tipografico.

- directory

  Vedere i86.1.1.

- inode

  Vedere i86.1.1.

- link ---> collegamento *

  – symbolic link ---> collegamento simbolico
  – hard link ---> collegamento fisico

- umask ---> maschera dei permessi

  La documentazione della shell Bash fa riferimento al comando '`umask`' come a quello che imposta la «maschera di creazione dei file» per i processi elaborativi. Tuttavia, utilizzando questa definizione si perde di vista il compito preciso di questa maschera: quello di eliminare alcuni permessi in modo predefinito.

• sticky (bit) ---> (bit) Sticky

In pratica, viene usato sempre con l'iniziale maiuscola in modo da abbinarlo facilmente agli altri «s-bit»: SUID, SGID e Sticky.

Quando *sticky* viene usato in altri contesti, si potrebbe tradurre come «adesivo».

• mode ---> modalità dei permessi

Evidentemente si fa riferimento ai 12 bit che definiscono i permessi di un file, lasciando da parte la proprietà dei file.

• permessi di accesso

Si tratta degli ultimi nove bit della modalità dei permessi, in cui si regolano proprio gli accessi a file e directory.

• mount, unmount ---> dipende dal contesto

– mount - -> innesto
– unmount - -> separazione
– mount point ---> punto di innesto
– directory di innesto
– to mount ---> innestare
– to unmount ---> staccare, separare

• home directory

La traduzione di questa definizione non è possibile in un modo unico, dal momento che si possono presentare situazioni differenti:

– ---> directory personale
quando si tratta di un utente umano, oppure quando si dà una personalità virtuale all'utente fittizio;
– ---> directory iniziale
quando si tratta di un utente fittizio riferito a un servizio, specialmente se questa directory è effettivamente l'«inizio» della gerarchia dell'applicativo (è evidente che questa definizione può essere usata solo se il contesto è compatibile).

• root ---> dipende dal contesto

– root directory ---> directory radice
– root file system ---> file system principale
– root partition ---> partizione principale

• path, pathname ---> percorso

I termini *path* e *pathname*, quando riguardano il percorso di un file o di una directory, hanno una differenza sottile che non sempre viene tenuta in considerazione nel modo corretto: il *pathname* dovrebbe essere un percorso che contiene l'informazione dell'oggetto finale (il file o la directory finale che si vuole indicare); il *path* dovrebbe essere il percorso della directory che contiene un oggetto a cui si fa riferimento.

A seconda dell'opportunità o meno, si può usare anche la forma «nome di percorso».

• percorso relativo

percorso assoluto

I due casi fanno riferimento rispettivamente a un percorso che parte dalla posizione di partenza e un percorso che parte invariabilmente dalla radice. In generale, la forma «percorso completo» è ambigua, perché può far pensare al *pathname*, pertanto è meglio evitarla.

• ramdisk, RAM disk ---> disco RAM

• backup ---> dipende dal contesto

La parola *backup* è il classico esempio di termine conciso e ambiguo della lingua inglese. Per tradurlo occorre utilizzare definizioni differenti a seconda del contesto. Segue un elenco di definizioni che potrebbero essere utilizzate a seconda del contesto particolare e a seconda del gusto del momento.

– copia di sicurezza, salvataggio
In questo caso si intende il *backup* come la copia che si fa per premunirsi contro le perdite di dati accidentali.
– copia di sicurezza di versioni precedenti
Alcuni programmi che copiano o spostano dei file, se incontrano altri file con lo stesso nome nella destinazione, cambiano il nome di questi ultimi, aggiungendo un'estensione simbolica (di solito una tilde, o il simbolo '#'). Queste sono delle copie di *backup*, nel senso che sono le copie di sicurezza delle versioni precedenti di quei file.
– copia di riserva
La copia di riserva è una copia che si affianca all'«oggetto» che si utilizza (il file, il dischetto, ecc.), nel caso questo risulti danneggiato.

• Linux native (partition) ---> (partizione) Linux-nativa *

• Linux swap (partition) ---> (partizione) Linux-swap

Utenza

«

• user ---> utente, utilizzatore

Vale la pena di distinguere tra l'utente inteso come entità che accede al sistema, rispetto all'utilizzatore (umano) di qualcosa.

• utente comune

L'utente comune dovrebbe essere inteso come l'utente di un sistema Unix che non ha privilegi particolari, ovvero un utente che non è l'amministratore (né '`root`', né un altro amministratore di qualche parte particolare del sistema).

• utilizzatore normale

L'utilizzatore normale dovrebbe essere quella persona che utilizza un accesso o un servizio senza grandi pretese e senza competenze speciali.

• utente normale

In alcuni casi, la definizione «utente comune» non va bene, per esempio quando si parla degli utenti normali del servizio WU-FTP.

• user name ---> nominativo-utente

Si tratta del nome che un utente utilizza per identificarsi e accedere al sistema. Al nominativo-utente si abbina una parola d'ordine.

• account ---> dipende dal contesto

Il termine *account* non è traducibile in un modo solo per tutti i contesti in cui si può usare in inglese. Segue un elenco di definizioni che potrebbero essere utilizzate a seconda del contesto particolare e a seconda del gusto del momento.

– utente -- quando si fa riferimento a un «utente logico» del sistema;
– utente registrato (nel sistema);
– utenza -- quando si vede l'aspetto contabile della faccenda, ovvero quando l'*account* è più vicino all'idea di un contratto per ottenere l'accesso;
– accesso;
– recapito -- nella posta elettronica;
– profilo (personale) -- quando si fa riferimento a un file di configurazione collocato nella directory personale;
– privilegi (di un certo utente) -- quando l'utente serve a fare o a evitare che sia fatto qualcosa di particolare;
– identità (di un utente).

• client, server ---> cliente, servente

I termini cliente e servente sono ambigui, sia in italiano che nell'originale inglese. Il problema nasce dal fatto che dipende dal contesto cosa sia «cliente» e cosa sia «servente». In un testo scritto in lingua italiana, dovrebbe essere auspicabile il chiarimento del contesto, come viene proposto nell'elenco seguente:

– programma cliente, programma servente
quando si fa riferimento a un programma che utilizza o che fornisce un servizio di qualche tipo;

– nodo cliente, nodo servente
quando si fa riferimento a una connessione in cui si distingue tra nodi che chiedono un servizio e nodi che forniscono un servizio, tenendo presente che all'interno dei nodi ci sono ovviamente dei programmi clienti e dei programmi serventi;

– elaboratore cliente, elaboratore servente
quando si fa riferimento all'elaboratore in cui si utilizza un programma cliente o un programma servente, senza voler porre un'enfasi particolare sul collegamento di rete.

Documentazione

«

- man page ---> pagina di manuale
  Lo Unix AT&T aveva un manuale cartaceo, diviso in sezioni, dove ogni comando costituiva una sottosezione. La composizione del manuale avveniva attraverso Troff ed era disponibile anche tramite il comando '**`man`**', abbreviazione di *manual*.
- on-line help ---> guida interna
  Si può considerare anche la possibilità di usare la forma «guida in linea», se appropriato.
- help ---> guida, guida interna

Interfaccia grafica

«

- desktop ---> superficie grafica ---> scrivania grafica
  A seconda del contesto, può essere più appropriata la definizione di superficie grafica, oppure di scrivania grafica. Per la precisione, la superficie dello schermo, quando viene usato con un gestore di finestre comune, è da intendersi semplicemente una superficie grafica, mentre un sistema più complesso (come Gnome) può essere definito come scrivania grafica.
- session manager ---> gestore di sessione
  Si tratta per esempio di Gnome o KDE, visti nell'ambito del controllo della sessione di lavoro con il sistema grafico X. Si parla di sessione quando si usa un *display manager*, come Xdm, Gdm, Kdm e simili.
- display manager ---> sistema grafico di autenticazione
  Si tratta per esempio di Xdm, Gdm, Kdm e simili.
- root window ---> finestra principale
  Utilizzando questa traduzione, occorre fare attenzione a non usare la stessa definizione per fare riferimento alla finestra più importante di un programma che può presentare diversi componenti su più finestre.
- screen saver ---> salva-schermo
- window manager ---> gestore di finestre
- stazione grafica
  X utilizza una definizione un po' contradditoria dei componenti di ciò che qui viene chiamato stazione grafica. Con questa definizione si fa riferimento al servizio offerto da un servente X; in tal modo, se ci sono più serventi X in funzione, ci sono altrettante stazioni grafiche virtuali, esattamente come accade per le console virtuali. In generale, X fa riferimento al *display* per indicare la stazione grafica, solo che poi, quando si tratta di indicare anche lo schermo, si utilizza l'opzione o la variabile di ambiente ***DISPLAY***, mentre in questo caso sarebbe opportuno parlare di «schermo» (*screen*) in modo preciso.
- pulsante grafico
  Quando si tratta di un tasto virtuale che appare sullo schermo.
- checkbox ---> casella di spunta

- mouse pointer, mouse cursor ---> puntatore del mouse
  Questo sembra essere un modo elegante per specificare che non si tratta del cursore all'interno del testo.[2]

Rete e comunicazioni

«

- datagram - -> datagramma
  Si tratta dei pacchetti di un protocollo non connesso (UDP).
- bridge
  Vedere i86.1.1.
- switch ---> commutatore di pacchetto *
  La traduzione non è diffusa, ma il termine originale è anche troppo generico.
- router
  Vedere i86.1.1.
- gateway
  Vedere i86.1.1.
- proxy
  Vedere i86.1.1.
- route ---> instradamento
- to route ---> instradare
- regola di instradamento *
  Una voce nella tabella degli instradamenti.
- Unix domain socket ---> socket di dominio Unix - -> socket di tipo Unix
  Meglio la prima delle due possibilità.
- to forward ---> inoltrare - -> proseguire
  In generale, «inoltrare» è la traduzione corretta, a parte una situazione particolare: nella posta tradizionale, quando una corrispondenza deve essere inviata a un indirizzo diverso da quello stabilito originariamente, questa «viene proseguita». Infatti, il problema si pone nel momento della consegna della corrispondenza: il postino viene a sapere che il destinatario ha cambiato indirizzo, oppure la stessa persona che l'ha ricevuta la reimbuca dopo aver modificato l'indirizzo di destinazione. Di conseguenza, sarebbe giusto dire che «si prosegue» un messaggio di posta elettronica quando questo, una volta giunto alla sua destinazione prevista, viene rinviato a un'altra destinazione.
- relay ---> relè *
- link (HTML) ---> riferimento, riferimento ipertestuale *, collegamento ipertestuale *
  In generale, i due termini, riferimento ipertestuale e collegamento ipertestuale, sono la stessa cosa. Eventualmente, a collegamento ipertestuale si può dare un'enfasi locale, mentre a riferimento ipertestuale un significato più lontano. In pratica, un riferimento interno a una stessa pagina HTML, o ad altre pagine che compongono un insieme ben organizzato, sarebbe un collegamento ipertestuale, mentre un riferimento a una risorsa esterna sarebbe un riferimento ipertestuale. Volendo evitare di fare confusione, conviene usare una definizione sola e precisamente riferimento ipertestuale.
- link (IPv6) ---> collegamento di rete
- computer host ---> elaboratore host, host ---> nodo di rete, nodo - -> stazione
  In questo caso si tratta di un elaboratore connesso in rete che in qualche modo ospita qualche servizio. Nel testo si preferisce usare il termine «nodo di rete» o soltanto «nodo».
  Il termine *host*, viene usato in particolare nella documentazione RFC riferita a IPv6 per indicare un nodo che non sia un router. Inoltre, sempre la terminologia riferita a IPv6 indica il nodo come qualunque dispositivo che utilizzi in pratica questo protocollo.

In italiano si utilizza anche il termine «stazione», seguito da un aggettivo che ne specifica il comportamento. Per esempio, nel capitolo dedicato alla realizzazione di elaboratori senza disco, si parla di stazioni senza disco.

- nodo di rete, nodo

  Quando si fa riferimento a un indirizzo nella rete, senza specificare il ruolo che ha ciò che vi corrisponde.

- diskless ---> senza disco

  Si fa riferimento a nodi di rete composti da elaboratori senza un disco locale da cui possa essere innestato il file system principale (la directory radice). Questi utilizzano il protocollo NFS per l'innesto di tutto il loro file system.

- netmask ---> maschera di rete (IPv4)

  Non vengono segnalate le abbreviazioni contenenti solo la parola «maschera».

- IP masquerading ---> mascheramento IP *

  La scelta di utilizzare il termine «mascheramento» come traduzione di *masquerading* in riferimento ai pacchetti IP, è discutibile. In generale, da un punto di vista logico, la traduzione corretta di questo termine dovrebbe essere «travestimento», o anche «camuffamento», dal momento che lo scopo del *masquerading* non è quello di nascondere i pacchetti, ma di farli sembrare appartenenti a un'origine differente. In questo documento si preferisce l'uso di «mascheramento», puntando sulla somiglianza letterale del termine con quello originale inglese, oltre al fatto che comunque si ottiene l'effetto di nascondere i nodi reali da cui hanno origine le comunicazioni.

- name server - -> servizio di risoluzione dei nomi *

  La traduzione fatta in questo modo cambia un po' il contesto: *name server* è un nodo che offre un servizio e non il servizio in sé. Quando si vuole fare riferimento proprio al nodo, si può parlare di servente DNS.

- root domain ---> dominio principale

  Il dominio di «primo livello» è quello che segue immediatamente quello principale; quindi, il dominio principale si rappresenta con un punto singolo, quando il contesto lo richiede, mentre il dominio di primo livello (che discende da quello principale), noto anche come TLD (*Top level domain*) potrebbe essere: *com, edu, net, org,...*

- packet driver ---> driver di pacchetto

  Si tratta del programma Dos utilizzato per comandare l'interfaccia di rete in modo da offrire ad altri programmi l'accesso alla stessa, attraverso un IRQ software.

- format prefix (IPv6) ---> prefisso di formato *

  Rappresenta l'idea di maschera di rete del sistema IPv6.

- interface identifier (IPv6) ---> identificatore di interfaccia

- group identifier (IPv6) ---> identificatore di gruppo

- mirror ---> sito speculare, riproduzione speculare

  Meglio la seconda delle due espressioni.

- mailing-list ---> lista di posta elettronica *, lista

- master ---> principale

  slave ---> secondario

  Questa traduzione va bene quando si tratta di serventi di qualche servizio, in cui uno solo è *master*, mentre tutti gli altri sono *slave*. Questa forma è stata usata in particolare per la descrizione del servizio NIS, nella sezione 36.4.

- master ---> primario

  slave ---> secondario

  Questa traduzione va bene quando si fa riferimento al servizio DNS, dal momento che in passato, il servente *master* veniva definito *primary*.

- chat script ---> script di chat ---> script di colloquio *

- ISP, provider ---> fornitore di accesso a Internet

  Dal momento che la definizione è estremamente lunga, quando il contesto è chiaro, si potrebbe abbreviare a «fornitore di accesso», o anche solo «fornitore».

- chain ---> punto di controllo *

  Si fa riferimento al firewall Linux, secondo i kernel 2.2.∗ e 2.4.∗, dove questo termine individua un punto di intercettazione dei pacchetti IP, allo scopo di applicarvi delle regole (direttive) che si traducono in obiettivi, ovvero nella sorte dei pacchetti stessi.

- internet superserver, internet service daemon ---> supervisore dei servizi di rete

  Si tratta praticamente di '**`inetd`**' o di '**`xinetd`**', senza fare riferimento in modo preciso a questo o quel programma.

Tipografia

«

- specie (alfabetica)

  Si tratta di una classificazione dei caratteri in base al tipo di linguaggio per cui sono fatti: latino, cirillico, greco,...

- family - -> famiglia di caratteri - -> stile

  Lo stile è una forma di classificazione estetica di un carattere, contrassegnato da un nome, come per esempio il Times. Il termine «stile» va bene fino a quando si resta all'interno di una stessa specie. Alle volte ci sono delle *font family* che si riferiscono a specie differenti, come il tipo Symbol, o Dingbats. La definizione «famiglia di caratteri» potrebbe andare bene nel caso si voglia mantenere la stessa ambiguità. Questa definizione, famiglia di caratteri, viene anche usata effettivamente, però bisogna ricordare che nel linguaggio tipografico tradizionale italiano, la «famiglia» si riferisce precisamente a un gruppo stilistico con piccole varianti rispetto allo stile a cui appartiene. Bisogna fare attenzione.

- serie, variante seriale

  La serie è la diversificazione formale di uno stesso stile alfabetico. All'interno di uno stile, una serie può essere una variante di forma: il tondo, il corsivo, il neretto,...

- forma

  La forma del carattere: il tondo contrapposto al corsivo, il chiaro contrapposto al neretto e altre varianti (inclinato, chiarissimo, nero, nerissimo, ecc.).

  – pendenza

    Un aspetto della forma del carattere: tondo contrapposto a inclinato.

  – tono

    Un aspetto della forma del carattere: dal chiarissimo al nerissimo.

  – width ---> larghezza

    Un aspetto della forma del carattere: dallo strettissimo al larghissimo.

- body size ---> corpo

  L'altezza del carattere.

- interlinea

  Tecnicamente è la distanza tra le righe che si aggiunge alla distanza minima in funzione del corpo del carattere utilizzato. Tuttavia, con questo termine si fa spesso riferimento alla distanza tra le basi di una riga e della successiva (dattilografia).

- foundry ---> fonderia

- serif ---> grazie, linee terminali

  In italiano, il termine si usa generalmente al plurale.

- sans serif ---> lineare

  Si tratta di uno stile senza grazie.

- collezione alfabetica

  La distinzione tra maiuscole e minuscole.

- font ---> fonte tipografica, fonte di caratteri ---> fonte ---> tipoplesso

  font ---> carattere ---> tipo di carattere ---> carattere tipografico, carattere da stampa

  Il termine *font* non corrisponde esattamente a qualcosa di ben definito nella tradizione della terminologia tipografica italiana, di conseguenza, la traduzione con il termine «fonte» e i suoi vari abbinamenti è solo una forma di derivazione dall'inglese, altrettanto ambigua. Il termine tipoplesso, sembrerebbe essere il più appropriato, solo che si tratta di qualcosa che risulterebbe incomprensibile ai più.

  La scelta di usare la definizione «tipo di carattere», con tutte le altre varianti, può essere motivata da un contesto non molto impegnato dal punto di vista dei problemi che riguardano la composizione tipografica. In generale, la sua semplicità rende più comprensibile il testo al lettore che non abbia già delle nozioni di tipografia.

- polizza

  L'assortimento completo di caratteri di un corpo determinato. Le polizze compongono il tipoplesso. Nella lingua francese, il termine «police» (polizza) si usa per tradurre il termine inglese *font*.

- scala di corpi

  L'insieme dei corpi in cui può essere reso un certo tipo di carattere.

- traslitterazione

  Traduzione da un alfabeto a un altro, lettera per lettera. Nella traslazione di un testo composto in cirillico traslitterato in carattere latino, l'alfabeto latino è il traslitterante e l'alfabeto cirillico è il traslitterato.

- character set ---> insieme di caratteri

  Da una discussione è emerso che dovendo scegliere tra «gruppo di caratteri» e «insieme di caratteri» è meglio la seconda forma per vari motivi fondati sulla teoria degli insiemi.[3]

- orientamento della stampa

  In questo modo si può identificare come si stampa su un foglio di carta.

  – portrait ---> verticale
  – landscape ---> orizzontale
  – sea-scape ---> rovesciato
  – up side down ---> sottosopra

- segnatura

  Il numero di fogli che compone un fascicolo nell'ambito di un sistema di rilegatura a filo. In pratica, i fogli stampati vanno piegati a metà e poi cuciti sulla piega, in modo da poter essere sfogliati.

- format ---> composizione

  Un documento viene «composto» tipograficamente. Il concetto di composizione si adatta anche per la stringa usata nelle funzioni come ***printf()*** del linguaggio C.

Unicode

«

- code point ---> punto di codifica

  Il simbolo dal punto di vista della codifica.

- code unit ---> unità di codifica

  L'unità di memoria utilizzata per la rappresentazione della codifica.

- CCS: Coded Character Set ---> insieme di caratteri codificato

  L'insieme di caratteri codificato attraverso un intero non negativo. L'insieme di caratteri universale è l'insieme di caratteri codificato di Unicode.

- CEF: Character Encoding Form ---> forma di codifica del carattere *

  Mappa di trasformazione tra l'insieme di caratteri codificato e le sequenze di unità di codifica.

- CES: Character Encoding Scheme ---> schema di codifica del carattere *

  Mappa di trasformazione tra le sequenze di unità di codifica e le sequenze di byte.

- TES: Transfer Encoding Syntax ---> sintassi di codifica per il trasferimento *

  Metodo di trasformazione reversibile di una codifica per il trasferimento dei dati.

- wide char ---> carattere esteso

- wide string ---> stringa estesa

- Insieme di caratteri universale

  L'insieme di caratteri universale è l'insieme di caratteri codificato di Unicode.

SGML/XML

«

- tag ---> marcatore

- well-formed ---> corretto formalmente

  well-formedness ---> correttezza formale

  La correttezza del documento riferita al DTD, viene definita «validità».

- name space ---> dominio applicativo * - -> dominio *

Grafica

«

- interleaved ---> interfogliato

- mirror ---> ribaltamento speculare

  Si fa riferimento al ribaltamento dell'immagine che si ottiene come se questa fosse posta davanti a uno specchio.

- offset ---> scostamento, scarto

  L'idea viene dal lavoro di ATO (*Amiga translators' organization*).

- despeckle ---> filtro mediano

- thumbnail ---> provino

  Questa traduzione va bene quando il contesto riguarda la selezione di un'immagine da un elenco di riduzioni, i «provini», come quelli che si fanno in fotografia.

- flood fill ---> campitura

- to flood fill ---> campire

Usenet

«

- feed

  Vedere i86.1.1.

- news

  Vedere i86.1.1.

- newsgroup ---> gruppo di discussione (di Usenet) - -> gruppo

  La definizione «gruppo di discussione» è quella più diffusa, anche se per alcuni potrebbe risultare imprecisa: non sempre si tratta di aree di discussione, potrebbero essere semplicemente dei gruppi per la diffusione di notizie di qualche tipo, senza che si formi una discussione vera e propria.

- news server, discussion host ---> servente di news

  Si tratta di un nodo di rete che offre l'accesso ad alcuni gruppi per mezzo del protocollo NNTP.

- to post ---> spedire (un articolo).

- sito Usenet

  Si tratta di un sito che offre un servizio di accesso alla rete Usenet.

- articolo

  L'articolo è ciò che viene diffuso attraverso Usenet, nei gruppi di discussione verso cui è stato spedito. Non si deve confondere con news, che invece rappresenta il servizio in generale.

Localizzazione

- collating sequence ---> sequenza di collazione

  L'insieme ordinato dei simboli (*collating element*) utilizzati in una localizzazione particolare.

- collating element ---> elemento di collazione

  Un elemento (un simbolo) di una sequenza di collazione.

- collating symbol ---> simbolo di collazione

  È il simbolo utilizzato per rappresentare un elemento di collazione nella localizzazione. Di solito si tratta di forme del tipo '**<a>**', '**<b>**', '**<c>**', ecc., come si vede nei file '`/usr/share/i18n/locales/*`'.

- equivalence class ---> classe di equivalenza

  Una classe di equivalenza identifica un gruppo di elementi di collazione (in certi casi si parla di caratteri equivalenti, ma si tratta generalmente di una scorciatoia giustificata solo dal contesto), che devono essere trattati come equivalenti per qualche motivo (di solito ai fini dell'ordinamento). Per esempio, le lettere «e», «è», «é» potrebbero essere trattate come equivalenti.

- character class ---> classe di caratteri

  Una classe di caratteri identifica un insieme dei caratteri attraverso un nome. Si distingue solitamente tra: lettere minuscole, lettere maiuscole, cifre numeriche, caratteri alfanumerici, ecc.

Varie

- maintainer ---> curatore
- contributor ---> collaboratore
- implementation ---> realizzazione - -> attuazione, adattamento
- to implement ---> realizzare - -> attuare, adattare
- keyword ---> parola chiave, parole chiave
- retry ---> tentativi ripetuti
- disclaimer ---> liberatoria
- flag ---> opzione (booleana), modalità (booleana), attributo (booleano), variabile (booleana), indicatore

  Purtroppo si possono tradurre in questo modo solo alcune situazioni.

- file manager ---> gestore di file.

  Si tratta di programmi come Midnight Commander, XFM e simili.

- login ---> accesso, procedura di accesso *
- logout ---> conclusione dell'accesso, conclusione della sessione di lavoro
- screen saver ---> salva-schermo
- hard limit, soft limit ---> limite fisico, limite logico
- lock ---> blocco
- signal trap ---> cattura di un segnale

- to prepend ---> anteporre

  Si fa riferimento all'aggiunta di qualcosa all'inizio di un flusso di dati, o all'inizio di un file.

- et al ---> et alia ---> e altri - -> e simili, ecc.

- menu ---> menù

  In generale, su alcuni vocabolari è ammesso l'uso del termine «menu» senza accento. Tuttavia, la norma UNI 6015 (47.3.1.4), fa espresso riferimento alle «parole polisillabe su cui la posa della voce cade sulla vocale che è alla fine della parola...».

- password ---> parola d'ordine.

  passphrase ---> parola d'ordine.

  Diventa difficile trovare una traduzione «perfetta» di questi due termini. Volendo tornare alle origini, la traduzione dovrebbe essere «parola d'ordine». Anche se non è un termine usato, rende l'idea.

  Nel caso particolare di *passphrase*, diventa impossibile una traduzione secondo il criterio indicato, se non perdendo l'informazione cruciale sulla lunghezza che la parola d'ordine deve avere, non essendo più una sola «parola».

  Va annotato comunque che esiste anche la forma «chiave di identificazione», nota almeno nei vocabolari. Si opta comunque per la traduzione originale anche perché il concetto di identificazione si può confondere con il nome fittizio abbinato a un utente.

- shadow password ---> parole d'ordine oscurate

- peso - -> massa

  Di solito si confonde il peso con la massa di un corpo. Il peso rappresenta una forza che si misura in newton (simbolo: «N»), mentre la massa si misura in kilogrammi (simbolo: «kg»).[4] Pertanto, quando si vuole rappresentare qualcosa che si esprime in multipli o sottomultipli del kilogrammo,[5] si fa riferimento a una massa.

Forme espressive particolari

- ridirezione

  È una questione di gusto personale, dal momento che molti preferiscono «re-direzione».[6]

- emettere attraverso lo standard output, emettere attraverso lo standard error

  Questa forma è quella usata nel documento. I motivi per cui è stata scelta sono tanti, ma non derivano da un'esperienza Unix. In generale, viene contestato che standard output e standard error sono file come gli altri, secondo la filosofia Unix, per cui su questi ci si «scrive».

Annotazioni varie

Le annotazioni che si fanno qui, non si riferiscono a forme usate nell'opera, ma si tratta comunque di qualcosa di interessante, eventualmente anche per un possibile uso futuro.

- produttività

  Questo termine potrebbe essere utilizzato al posto di «velocità», quando si fa riferimento alla quantità di dati che possono transitare nell'unità di tempo. In altri termini, invece di parlare di velocità di un modem, si potrebbe parlare di produttività.

- ricorrente

  ricorrenza

  In matematica, si preferisce usare il termine «ricorrente» al posto di «ricorsivo» e «ricorrenza» al posto di «ricorsione», ma in informatica, questa forma (ormai desueta) fa pensare alle iterazioni pure e semplici.

Nomi dei caratteri speciali

La tabella u86.9 elenca alcuni caratteri e simboli speciali, assieme alla denominazione usata in questo documento.

Tabella u86.9. Elenco dei nomi di alcuni caratteri e altri simboli.

Simbolo	Denominazione
-	trattino (normale)
–	trattino basso
|	barra verticale
/	barra obliqua (normale)
\	barra obliqua (inversa)
'	apice singolo
‘	apice inverso
"	apice doppio, virgolette, virgolette alte
« »	virgolette basse, virgolette uncinate
&	e-commerciale
~	tilde
@	*at*, chiocciola, chiocciolina, chioccioletta -- meglio non usarlo
#	cancelletto -- meglio non usarlo
:	due punti (verticali)
..	due punti in orizzontale

In particolare, i simboli elencati di seguito meritano maggiore attenzione.

- @
  In origine questo simbolo è nato per abbreviare la parola latina «ad», mentre oggi si conosce prevalentemente la sua traduzione inglese: *at*. Sembra ricorrente il nome «chiocciola» in italiano, ma in generale non è il caso di nominarla in un testo scritto.
- #
  È difficile dare un nome a questo simbolo; attualmente è diffuso il termine «cancelletto» nel settore della telefonia, mentre è noto l’uso che se ne fa nell’ambito musicale, a rappresentare un diesis.

Nomi da usare in modo uniforme

Per molto tempo nell’opera è stato usato l’elemento ‘**`special`**’, con attributo ‘**`name`**’ per annotare e ricordare l’uso di nomi ricorrenti, da usare in modo coerente, soprattutto per ciò che riguarda la scelta di maiuscole e minuscole. Per quei nomi a cui questo meccanismo non si applica o non si applica più, viene conservata la tabella successiva.

Tabella u86.10. Nomi da usare in modo uniforme nel testo discorsivo.

Nome	Annotazioni
C	Linguaggio di programmazione C.
C++	Linguaggio di programmazione C++.
GNU C	Compilatore C del progetto GNU.
GNU AS	Assemblatore del progetto GNU, noto anche con il nome GAS.
NASM	Assemblatore specifico per codice Intel.

Riferimenti

- *Amiga Translators’ Organization*
  http://bilbo.di.unipi.it/~ato-it/
- Silvano Gai, *IPv6*, McGraw-Hill, 1997, ISBN 88-386-3209-X
- Bureau International des Poids et Mesures, *Le Système international d’unités (SI)*
  http://www1.bipm.org/utils/en/pdf/brochure-si.pdf
- Bureau International des Poids et Mesures, *The International System of Units (SI)* (traduzione in inglese)
  http://www1.bipm.org/utils/en/pdf/si-brochure.pdf
- National Institute of Standards and Technology, *International System of Units (SI)*
  http://physics.nist.gov/cuu/Units/index.html
- National Institute of Standards and Technology, *Guide for the Use of the International System of Units (SI)*, 1995
  http://physics.nist.gov/cuu/pdf/sp811.pdf
- Markus Kuhn, *Standardized Units for Use in Information Technology*, 1995
  http://www.cl.cam.ac.uk/~mgk25/information-units.txt
- National Institute of Standards and Technology, *Prefixes for binary multiples*
  http://physics.nist.gov/cuu/Units/binary.html
- *Grafica; scienza, tecnologia e arte della stampa e della comunicazione*, Arti poligrafiche europee
  http://www.apenet.it/

Indice del glossario stilistico

accesso 587 594 account 587 adapter 579 adattamento 585 594 adattare 585 594 adattatore 579 adattatore grafico 579 adattatore SCSI 579 alloggiamento 579 anteporre 594 applicazione concorrente 577 applicazione distribuita 577 applicazione multithread 577 applicazione parallela 577 archive 581 archiviazione 581 archivio 581 archivio compresso 581 archivio delle registrazioni 578 array 574 584 array associativo 584 arresto del sistema 578 articolo 594 assegnamento 584 associative array 584 associazione dei tasti 581 attributo 594 attuare 594 attuazione 594 avvio 578 background 577 backup 586 back-end 578 base di dati 582 basi di dati 582 bit rate 583 blocco 594 body size 591 boot 578 bridge 574 589 buffer 579 cache memory 579 campire 593 campitura 593 campo di controllo 582 carattere 592 carattere da stampa 592 carattere di controllo 582 carattere di tabulazione 580 carattere esteso 593 carattere interrupt 581 carattere tipografico 592 caricamento 578 carico 582 casella di spunta 588 cattura di un segnale 594 CEST 576 CET 576 576 chain 591 character class 594 Character Encoding Form 593 Character Encoding Scheme 593 character set 592 chat script 590 checkbox 588 checksum 582 cifrato 583 583 cifratura 583 583 cifra di controllo 582 cipher 583 classe di caratteri 594 classe di equivalenza 594 client 587 cliente 587 coda 578 Coded Character Set 592 code point 592 code unit 592 codice di controllo 582 582 codice di EOF 580 codice di escape 580 codice di interruzione di riga 580 collaboratore 594 collating element 594 collating sequence 594 collating symbol 594 collegamento 585 collegamento di rete 589 collegamento fisico 585 collegamento ipertestuale 589 collegamento simbolico 585 collezione alfabetica 591 Common Name 583 commutatore di pacchetto 589 composizione 585 592 computer 579 computer host 589 conclusione della sessione di lavoro 594 conclusione dell’accesso 594 condotto 577 congiunzione 582 contributor 594 controller 579 controllo 582 conversion specifier 584 copia di riserva 587 copia di sicurezza 586 copia di sicurezza di versioni precedenti 586 corpo 591 correttezza formale 593 corretto

formalmente 593 crittografia 583 curatore 594 database 582 582 datagram 589 datagramma 589 data type 582 daylight saving time 576 decrittazione 583 desktop 588 despeckle 593 device 579 device driver 579 device file 579 device number 579 directory 574 585 directory di innesto 586 directory iniziale 586 directory radice 586 directory personale 586 disclaimer 594 disco fisso 585 disco rigido 585 disco RAM 586 discussion host 593 diskless 590 display manager 588 dispositivo 579 dispositivo fisico 579 dispositivo logico 579 Distinguishing Name 583 dominio 593 dominio applicativo 593 dominio principale 590 download 582 driver 579 579 driver di pacchetto 590 ecc. 594 elaboratore 579 elaboratore cliente 588 elaboratore host 589 elaboratore servente 588 elemento di collazione 594 elenco 582 emettere attraverso lo standard error 6-595 emettere attraverso lo standard output 6-595 empty string 582 encrypted 583 encryption 583 eof 580 EOF 580 equijoin 582 equivalence class 594 equi-giunzione 582 equi-giunzione incompleta 582 escape 580 espressione regolare 581 estrazione 581 et al 594 et alia 594 exit status 578 e altri 594 e simili 594 famiglia di caratteri 591 family 591 feed 574 593 filehandle 585 file-make 585 file delle registrazioni 578 file di differenze 581 file di dispositivo 579 file di protezione 578 file di registrazioni 578 file lucchetto 578 file manager 594 file per il controllo dell'accesso 578 file per le registrazioni 578 file handle 585 file system principale 586 filtro mediano 593 finestra principale 588 firma MD5 582 flag 594 flood fill 593 flusso 584 585 flusso di file 585 fonderia 591 font 592 592 fonte 592 fonte di caratteri 592 fonte tipografica 592 foreground 577 forma 591 format 585 585 592 formattazione 585 format prefix 590 forma di codifica del carattere 593 fornitore di accesso a Internet 590 foundry 591 front-end 578 fuso 576 fuso orario 576 gateway 574 589 gestione 579 gestire 580 gestore 579 gestore di dispositivo 579 gestore di file 594 gestore di finestre 588 gestore di sessione 588 giunzione 582 grazie 591 grilletto 582 group identifier 590 gruppo 593 gruppo di discussione 593 gruppo di elaborazione 578 guida 588 guida interna 588 588 hard disk 585 hard limit 594 hard link 585 help 588 home directory 586 host 589 identificatore di gruppo 590 identificatore di interfaccia 590 identità 587 implementation 594 indicatore 594 informazione data-orario 576 Init 578 578 inizializzazione 585 innestare 586 innesto 586 inode 574 585 inoltrare 589 insieme di caratteri 592 insieme di caratteri codificato 592 Insieme di caratteri universale 593 instradamento 589 instradare 589 interfaccia SCSI 579 interface identifier 590 interfogliato 593 interleaved 593 interlinea 591 intermediario 575 intermediazione 575 internet service daemon 591 internet superserver 591 interrupt 578 interrupt character 581 interruzione 578 invito 576 in chiaro 583 in cifra 583 in monoprogrammazione 577 in multiprogrammazione 577 in primo piano 577 ISP 590 join 582 keyword 594 key binding 581 landscape 592 larghezza 591 liberatoria 594 limite fisico 594 limite logico 594 lineare 591 linee terminali 591 linguaggio concorrente 577 linguaggio di programmazione concorrente 577 linguaggio di script 584 linguaggio script 584 link 585 589 589 Linux-nativa 587 Linux-swap 587 Linux native 587 Linux swap 587 lista 590 lista di posta elettronica 590 livello 578 livello di esecuzione 578 lock 594 lock file 578 log 578 login 594 logout 594 log archive 578 log file 578 magic number 574 581 mailing-list 590 maintainer 594 major number 579 makefile 585 man page 588 marcatore 593 mascheramento 590 maschera dei permessi 585 maschera di rete 590 masquerading 590 massa 595 master 590 590 MD5 digest 582 MD5 message digest 582 memoria cache 574 579 memoria non volatile 579 memoria tampone 579 menu 595 menù 595 messaggio del giorno 581 messaggio di pubblicazione 581 MET 576 minor number 579 mirror 590 593 modalità 594 modalità dei permessi 586 mode 586 monoprogrammato 577 monoprogrammazione 577 mount 586 586 mount point 586 mouse cursor 588 mouse pointer 588 multielaborazione 577 multiprogrammato 577 multiprogrammazione 577 multitasking 577 name server 590 name space 593 netmask 590 news 574 593 newsgroup 593 news server

593 new-line 580 nodo 589 590 nodo cliente 588 nodo di rete 589 590 nodo servente 588 nome comune 583 nome distintivo 583 nominativo-utente 587 numero di dispositivo 579 numero primario 579 numero secondario 579 nvram 579 octet 582 offset 593 on-line help 588 opzione 594 ora estiva 576 orientamento 592 orizzontale 592 ottetto 582 outer-join 582 overflow 582 pacchetto 581 package 581 packet driver 590 pagina di manuale 588 parametro 584 parametro formale 584 parola chiave 594 parola d'ordine 595 595 parole chiave 594 parole d'ordine oscurate 595 parte frontale 578 parte terminale 578 partizione principale 586 passphrase 595 password 595 patch 581 path 586 pathname 586 pendenza 591 percorso 586 percorso assoluto 586 percorso relativo 586 permessi di accesso 586 peso 595 ping 574 pipe 577 pipeline 577 pixel 575 polizza 592 porting 585 portrait 592 precompilatore 584 prefisso di formato 590 preprocessor 584 primario 590 principale 590 print job 578 privilegi 587 procedura di accesso 594 procedura di arresto del sistema 578 procedura di avvio del sistema 578 procedura di inizializzazione del sistema 578 processo di stampa 578 processo iniziale 578 produttività 6-595 profilo 587 programmazione concorrente 577 programma cliente 587 programma di servizio 577 programma di utilità 577 programma frontale 578 programma sequenziale 578 programma servente 587 programma terminale 578 programmi di utilità 577 prompt 576 proseguire 589 provider 590 provino 593 proxy 575 589 pulsante grafico 588 puntatore del mouse 588 punto di codifica 592 punto di controllo 591 punto di innesto 586 punto grafico 575 ramdisk 586 RAM disk 586 realizzare 594 realizzazione 594 recapito 587 record 575 581 registrare 578 registrazione degli eventi 578 registro 578 582 registro del sistema 578 registro elettronico 578 regola di instradamento 589 regular expression 581 relay 589 relè 589 retry 594 ribaltamento speculare 593 ricorrente 6-595 ricorrenza 6-595 ridirezione 6-595 riferimento 589 riferimento ipertestuale 589 riga di comando 576 riproduzione speculare 590 root 586 root directory 586 root domain 590 root file system 586 root partition 586 root window 588 route 589 router 574 589 rovesciato 592 runlevel 578 salvataggio 586 salva-schermo 588 594 sans serif 591 scala di corpi 592 scambio 579 scarico 582 scarto 593 scheda di controllo 579 scheda grafica 579 scheda SCSI 579 scheda video 579 schema di codifica del carattere 593 scostamento 593 screen saver 588 594 script 575 584 scripting language 584 script di chat 590 script di colloquio 590 script language 584 scrivania grafica 588 sea-scape 592 secondario 590 590 segnatura 592 senza disco 590 separare 586 separazione 586 sequenza di collazione 594 sequenza di escape 580 serie 591 serif 591 servente 587 servente di news 593 server 587 servizio di risoluzione dei nomi 590 session manager 588 shadow password 595 shell job 578 shutdown 578 signal trap 594 simbolo di collazione 594 singletasking 577 sintassi di codifica per il trasferimento 593 sistema 579 sistema di elaborazione 579 sistema grafico di autenticazione 588 sito speculare 590 sito Usenet 594 slave 590 590 slot 579 socket di dominio Unix 589 socket di tipo Unix 589 soft limit 594 somma di controllo 582 sottosopra 592 specie 591 specificatore di conversione 584 spedire 593 spool 578 staccare 586 stack 575 standard error 575 581 standard input 575 581 standard output 575 581 stazione 589 stazione grafica 588 Sticky 585 sticky 585 stile 591 stream 584 stringa estesa 593 stringa nulla 582 582 stringa vuota 582 sullo sfondo 577 superficie grafica 588 supervisore dei servizi di rete 591 swap 579 switch 589 symbolic link 585 system log 578 tab 580 tabella 582 tag 593 task 575 577 tasso del flusso di dati 583 tempo medio dell'europa centrale 576 tempo universale 576 tempo universale coordinato 576 tentativi ripetuti 594 terminale a caratteri 579 terminali a caratteri 579 thumbnail 593 timestamp 576 time zone 576 tipi di dati 582 tipoplesso 592 tipo di carattere 592 tipo di dati 582 tono 591 to drive 580 to flood fill 593 to forward 589 to implement 594 to log 578 to mount 586 to port 585 to post 593 to prepend 594 to route 589 to unmount 586 traboccare 582 583 Transfer Encoding Syntax 593 traslitterazione 592 trigger 582 umask 585 underflow 583 unità

di codifica 592 unità di controllo 579 universal time 576 universal time coordinated 576 Unix domain socket 589 unmount 586 586 upload 582 up side down 592 user 587 user name 587 UT 576 UTC 576 utente 587 587 utente comune 587 utente normale 587 utente registrato 587 utenza 587 utility 577 577 utilità 577 utilizzatore 587 utilizzatore normale 587 valore di uscita 578 variabile 594 variante seriale 591 verticale 592 well-formed 593 well-formedness 593 wide char 593 wide string 593 width 591 window manager 588 zone 576 # 6-596 @ 6-596

[1] Anche la definizione «ora solare» è imprecisa, perché l'ora solare vera e propria non è la stessa su tutto il fuso orario a cui viene invece applicata

[2] Potrebbe essere interessante anche l'idea di «mirino» del mouse.

[3] Unicode introduce una terminologia più precisa al riguardo di ciò che un tempo si chiamava *character set*.

[4] 1 N = 1 kg*m/s^2

[5] 1 g = 10^{-3} kg

[6] Il termine «ridirezione» viene usato anche in *IPv6* di Silvano Gai, McGraw Hill, 1997, alla sezione 6.4.3, anche se in questo caso si tratta di ridirezione dei pacchetti IPv6.

Texinfo: lo standard della documentazione GNU

Introduzione a Texinfo 603
- Esempio introduttivo 603
- Logica fondamentale di Texinfo 604
- Struttura di un documento Texinfo 606
- Indici 609
- Aspetto del testo e ambienti speciali 612
- Elenchi e tabelle 614
- Modifica dello stile TeX 616
- Localizzazione 616
- Riferimenti 616

Texinfo: libro e ipertesto 617
- Sequenza dei nodi secondo Texinfo 618
- Definizione automatica della sequenza dei nodi e problemi relativi 619
- Limitazioni originali della struttura a nodi 620
- Riferimenti ipertestuali e limitazioni verbali 620
- Altri tipi di riferimento 622
- Riepilogo dei comandi relativi a nodi, ancore e riferimenti 623

Sgmltexi: installazione e utilizzo 625
- Installazione di Sgmltexi 625
- Come si usa il programma frontale 626
- Riferimenti 628

Sgmltexi: struttura 629
- Struttura generale per un sorgente Sgmltexi 629
- Scomposizione del documento, nodi e menù Info 638
- Codifica 640

Sgmltexi: contenuti 641
- Paragrafi 641
- Indici e riferimenti incrociati 641
- Delimitazione di parole e di frasi 643
- Delimitazione di blocchi di testo 644
- Elenchi e tabelle 645
- Inserzioni 647
- Definizioni 647
- Codice condizionato e codice letterale in base alla composizione 649

Corrispondenza tra Texinfo e Sgmltexi 653

Introduzione a Texinfo

Esempio introduttivo 603
Logica fondamentale di Texinfo 604
Scomposizione del documento 605
Inserimento di simboli speciali 605
Struttura di un documento Texinfo 606
Capitoli e sezioni 607
Nodi 608
Menù di riferimenti 609
Indici 609
Indice generale 610
Indici analitici 610
Definizione di indici aggiuntivi 612
Fusione di indici 612
Aspetto del testo e ambienti speciali 612
Elenchi e tabelle 614
Elenco puntato e numerato 614
Elenco descrittivo 614
Tabelle vere e proprie 615
Modifica dello stile TeX 616
Localizzazione 616
Riferimenti 616

Texinfo è un sistema di composizione ideato per la documentazione del progetto GNU, allo scopo di permettere la produzione di documenti ipertestuali in formato Info e di documenti stampati, attraverso il sistema di composizione TeX, a partire da un sorgente unico. Attualmente è disponibile anche la possibilità di comporre in HTML, cosa che completa il sistema Texinfo e lo rende uno strumento essenziale, ma anche molto valido.

A seconda di come è organizzata la propria distribuzione GNU, gli script che compongono il sistema Texinfo potrebbero far parte di un pacchetto indipendente, oppure essere inseriti direttamente all'interno della distribuzione teTeX (LaTeX).

Emacs permette di gestire in modo automatico molte particolarità del sorgente Texinfo, facilitando così il lavoro dell'utilizzatore. In questo capitolo si vuole mostrare solo l'essenziale di Texinfo, pertanto, tutta la parte che riguarderebbe la gestione di Emacs viene ignorata. Questo e altri particolari possono essere approfonditi nella documentazione originale di Texinfo.

Esempio introduttivo

Di solito, il modo migliore per cominciare a comprendere il funzionamento di un sistema di composizione, è quello di partire da un esempio, per avere modo di vedere subito come comporlo in pratica.

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename esempio.info
@settitle Introduzione a Texinfo
@c %**end of header

@setchapternewpage odd

@ifinfo
Questo è un esempio molto breve di un documento scritto
utilizzando il sistema Texinfo.

Copyright @copyright{} 1999 Tizio Tizi
@end ifinfo

@titlepage
@sp 10
@comment Per il titolo viene utilizzato un corpo molto grande.
@center @titlefont{Titolo di esempio}

@c I due comandi seguenti iniziano la pagina del copyright.
```

```
@page
@vskip 0pt plus 1filll
Copyright @copyright{} 1999 Tizio Tizi
@end titlepage

@node    Top, Suddivisione del documento, , (dir)
@comment nodo-attuale, nodo-successivo, nodo-precedente, nodo-superiore

@menu
* Suddivisione del documento::      Il primo capitolo di questo esempio
                                    molto breve.
* Paragrafi::                       Il secondo capitolo.
* Indice analitico::                L'indice analitico.
@end menu

@node    Suddivisione del documento, Paragrafi, Top, Top
@comment nodo-attuale, nodo-successivo, nodo-precedente, nodo-superiore
@chapter Suddivisione del documento con Texinfo
@cindex suddivisione
@cindex capitolo
@cindex sezione
@cindex sottosezione

Un documento scritto in Texinfo è organizzato in capitoli, che possono
essere suddivisi in sezioni, sottosezioni e sotto-sottosezioni:

@enumerate
@item
capitolo -- @code{@@chapter};
@item
sezione -- @code{@@section};
@item
sottosezione -- @code{@@subsection};
@item
sotto-sottosezione -- @code{@@subsubsection};
@end enumerate

@node    Paragrafi, Indice analitico, Suddivisione del documento, Top
@comment nodo-attuale, nodo-successivo, nodo-precedente, nodo-superiore
@chapter Paragrafi in un sorgente Texinfo
@cindex paragrafo
@cindex testo

Il testo normale di un documento scritto in Texinfo è suddiviso in
paragrafi senza l'indicazione esplicita di alcun comando speciale.
Di conseguenza, basta inserire una riga vuota nel sorgente, per
produrre la separazione tra un paragrafo e il successivo.

@node    Indice analitico, , Paragrafi, Top
@comment nodo-attuale, nodo-successivo, nodo-precedente, nodo-superiore
@unnumbered Indice analitico

@printindex cp

@contents
@bye
```

Si suppone di avere nominato il file di questo sorgente '`esempio.texinfo`'. Di seguito vengono mostrati i comandi necessari alla composizione per generare un file Info, un risultato in HTML (in due modi differenti), un file PostScript e un file PDF.

```
$ makeinfo esempio.texinfo [Invio]

$ makeinfo --html esempio.texinfo [Invio]

$ texi2html esempio.texinfo [Invio]

$ texi2dvi esempio.texinfo ; dvips -t a4 -o esempio.ps
esempio.dvi [Invio]

$ texi2dvi --pdf esempio.texinfo [Invio]
```

Nel primo caso viene generato il file Info '`esempio.info`'; nel secondo e nel terzo si ottiene il file '`esempio.html`' (affiancato eventualmente da un file contenente l'indice generale); nel quarto caso si ottiene il file '`esempio.ps`'; nell'ultimo si ottiene il file '`esempio.pdf`'.

Logica fondamentale di Texinfo

Texinfo è TeX a cui è stato applicato uno stile speciale, per cui il simbolo '`@`' sostituisce la barra obliqua inversa ('`\`'). Questo si ottiene attraverso uno stile contenuto nel file '`texinfo.tex`', che viene incluso opportunamente con il comando TeX iniziale:

```
\input texinfo
```

Da quel punto in poi, la barra obliqua inversa ha valore letterale. Sempre allo scopo di ridurre al minimo i simboli che hanno significati speciali, i commenti si indicano attraverso un comando apposito: '**`@c`**', oppure '**`@comment`**'. A questo proposito, si può osservare che la prima riga mostrata nell'esempio introduttivo, contiene proprio un commento, subito dopo la dichiarazione dell'inclusione dello stile per Texinfo:

```
\input texinfo    @c -*-texinfo-*-
```

Si tratta di una stringa convenzionale, che è bene utilizzare anche se non è strettamente necessaria alla composizione di un sorgente Texinfo, perché riguarda Emacs, permettendogli di identificare il file e di qualificarlo per quello che è.

Una volta chiarita la natura TeX di un sorgente Texinfo, si può comprendere il comportamento generale del sistema, nel momento in cui la composizione viene fatta per arrivare alla stampa. In particolare, si può intendere il modo in cui vengono considerati gli spazi, che vengono eliminati quando sembrano superflui, così come si può intendere il motivo per cui basta separare i blocchi di testo con una o più righe vuote (o bianche), per ottenere la separazione in paragrafi. Tuttavia, le cose cambiano quando la composizione avviene in modo da generare un file Info: in questo caso gli spazi aggiuntivi contano e anche le righe vuote superflue possono essere prese in considerazione.

Nonostante la sua natura TeX, Texinfo è orientato alla generazione di un ipertesto consultabile attraverso un terminale a caratteri; pertanto, è su questo punto che si fondano le sue caratteristiche e le sue limitazioni.

Scomposizione del documento

Un documento Texinfo è articolato in due modi distinti, che devono avvenire simultaneamente. Da una parte si trova l'articolazione del testo nel modo più adatto a un libro, con i suoi capitoli e le sezioni a livelli diversi (come fa LaTeX), dall'altra parte c'è un ipertesto organizzato a grafo (un reticolo di collegamenti uniti assieme da dei nodi), dove i nodi sono i vari blocchi di informazioni.

Texinfo non pone limitazioni particolari all'uso dei nodi, tuttavia il buon senso richiede che siano usati in modo compatibile con la struttura «cartacea» del documento. In generale, ogni capitolo deve avere un nodo corrispondente, mentre le sezioni potrebbero averlo se ciò è opportuno, e lo stesso vale per le sottosezioni. Nell'esempio introduttivo, prima della dichiarazione del primo capitolo, si vede l'indicazione del nodo relativo:

```
@node    Suddivisione del documento, Paragrafi, Top, Top
@comment nodo-attuale, nodo-successivo, nodo-precedente, nodo-superiore
@chapter Suddivisione del documento con Texinfo
```

Dal momento che la composizione in formati finali diversi genera risultati differenti, c'è poi l'esigenza di poter distinguere il testo che deve essere usato per una o l'altra composizione. Per questo si possono circoscrivere delle porzioni di testo tra i comandi '**`@ifinfo`**' '**`@end ifinfo`**', '**`@iftex`**' '**`@end iftex`**', e '**`@ifhtml`**' '**`@end ifhtml`**'. Nell'esempio introduttivo si vede proprio l'uso di questi comandi per inserire del testo che viene utilizzato solo nella composizione in formato Info:

```
@ifinfo
Questo è un esempio molto breve di un documento scritto
utilizzando il sistema Texinfo.

Copyright @copyright{} 1999 Tizio Tizi
@end ifinfo
```

Inserimento di simboli speciali

Il linguaggio di composizione utilizzato da Texinfo, attribuisce un significato speciale al simbolo '`@`' e alle parentesi graffe. Per indicare questi caratteri in modo letterale, basta farli precedere da un altro '`@`'. In questo senso, la sequenza '***`@carattere`***' rappresenta spesso la richiesta esplicita di fare riferimento al carattere in modo letterale. La tabella u87.6 elenca alcune di queste sequenze di escape.

Tabella u87.6. Comandi per rappresentare alcuni simboli speciali.

Comando	Descrizione
`@@`	Un solo simbolo '@'.
`@{`	Una parentesi graffa aperta.
`@}`	Una parentesi graffa chiusa.
`@<SP>`	Uno spazio non interrompibile.

In modo simile si possono definire delle lettere speciali, se queste non sono disponibili attraverso la tastiera. La tabella u87.7 mostra i comandi utili per rappresentare le vocali accentate italiane. Tuttavia, è opportuno osservare che non è sempre conveniente l'uso di questi comandi, se si ritiene di poter usare una codifica migliore dell'ASCII tradizionale. Infatti, se si usa un comando come '**`@‘a`**' si rischia poi di vedere '**`a‘`**' nella composizione Info, cosa che non succede se si utilizza la codifica ISO 8859-1.

Tabella u87.7. Comandi per la rappresentazione delle vocali accentate nella lingua italiana.

Comando	Risultato	Comando	Risultato
`@‘a`	à	`@‘A`	À
`@‘e`	è	`@‘E`	È
`@’e`	é	`@’E`	É
`@‘i`	ì	`@‘I`	Ì
`@‘o`	ò	`@‘O`	Ò
`@‘u`	ù	`@‘U`	Ù

Struttura di un documento Texinfo

In un sorgente Texinfo, prima di arrivare alla scomposizione del testo in capitoli e nodi, c'è una parte iniziale che merita un po' di attenzione. La prima dichiarazione in assoluto è quella dell'inserimento dello stile '`texinfo.tex`', come è già stato mostrato, quindi si incontrano le dichiarazioni '**`@setfilename`**' e '**`@settitle`**' che costituiscono l'intestazione del sorgente:

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename esempio.info
@settitle Introduzione a Texinfo
@c %**end of header
```

La prima di queste due dichiarazioni serve a definire il nome del file Info finale, che in caso di necessità potrebbe anche essere scomposto in più file, dove quello indicato rappresenta così solo il file di partenza; la seconda dichiara il titolo del documento in breve. Texinfo richiede che le dichiarazioni dell'intestazione siano racchiuse tra due commenti ben definiti, come si vede dall'esempio. È importante che siano riprodotti nello stesso modo che è stato mostrato:

```
@c %**start of header
...
@c %**end of header
```

La parte successiva all'intestazione, viene usata per utilizzare dei comandi specifici per la composizione TeX, come nel caso di '**`@setchapternewpage`**' che permette di definire se i capitoli debbano iniziare in una pagina nuova e se questa debba essere una pagina dispari, oppure se ciò sia indifferente:

Comando	Descrizione
`@setchapternewpage on`	fa in modo che ogni capitolo inizi a pagina nuova;
`@setchapternewpage off`	fa in modo che i capitoli possano iniziare nella stessa pagina in cui finiscono quelli precedenti;
`@setchapternewpage odd`	fa in modo che ogni capitolo inizi in una nuova pagina dispari.

Dopo questo genere di definizioni, si passa normalmente alla presentazione formale del documento, annotando le informazioni legali e, nel caso particolare della composizione in TeX, specificando anche l'aspetto della prima pagina, quella del titolo. Nel caso di un documento Info non ha molta importanza la preparazione di una facciata introduttiva; in effetti, questa non esiste (come è possibile vedere in seguito a proposito del nodo iniziale). In questo senso, non c'è nemmeno il posto per le informazioni sul copyright, che vengono comunque inserite, delimitandole tra i comandi '**`@ifinfo`**' e '**`@end ifinfo`**', allo scopo che queste siano effettivamente annotate all'inizio nel file Info, anche se in una zona che poi non viene consultata attraverso la navigazione ipertestuale.

```
@ifinfo
Questo è un esempio molto breve di un documento scritto
utilizzando il sistema Texinfo.

Copyright @copyright{} 1999 Tizio Tizi
@end ifinfo
```

Alla fine di questa parte introduttiva del sorgente Texinfo, appare generalmente un blocco delimitato dai comandi '**`@titlepage`**' e '**`@end titlepage`**', che riguardano esclusivamente la composizione con TeX, con lo scopo di definire le pagine iniziali dal titolo alle informazioni sul copyright.

```
@titlepage
@sp 10
@comment Per il titolo viene utilizzato un corpo molto grande.
@center @titlefont{Titolo di esempio}

@c I due comandi seguenti iniziano la pagina del copyright.
@page
@vskip 0pt plus 1filll
Copyright @copyright{} 1999 Tizio Tizi
@end titlepage
```

I comandi che si vedono nell'esempio dovrebbero essere abbastanza intuitivi, dal momento che si tratta praticamente di TeX:

Comando	Descrizione
`@sp` ***n***	richiede uno spazio verticale di ***n*** righe;
`@center @titlefont{` ***titolo*** `}`	centra il titolo che viene stampato utilizzando un carattere adatto, pensato appositamente per questo ('**`@titlefont`**');
`@page`	esegue un salto pagina;
`@vskip 0pt plus 1filll`	porta il testo successivo alla base della pagina (si tratta di un comando TeX, dove la parola '**`1filll`**' è scritta correttamente con tre «l»).

Il manuale di Texinfo propone anche un'altra forma, in cui si utilizzano i comandi '**`@title`**', '**`@subtitle`**' e '**`@author`**'. Il loro significato è intuitivo e l'esempio seguente dovrebbe chiarirne l'uso: si osservi in particolare la presenza di due sottotitoli e di due autori.

```
@titlepage
@title Titolo di esempio
@subtitle Primo sottotitolo
@subtitle Secondo sottotitolo
@author Tizio Tizi
@author Caio Cai
@page
@vskip 0pt plus 1filll
Copyright @copyright{} 1999 Tizio Tizi, Caio Cai
@end titlepage
```

Nell'esempio introduttivo che è stato mostrato, non appare circoscritto alcun pezzo riservato alla composizione in HTML. In effetti, '**`texi2html`**' ignora l'inizio del sorgente Texinfo, a parte l'intestazione, dalla quale ottiene il titolo del documento e il nome del file HTML principale che deve generare.

Al termine di un documento Texinfo, deve essere usato il comando '**`@bye`**' per concludere esplicitamente la composizione.

Capitoli e sezioni

Prima di affrontare il problema che riguarda la scomposizione del documento in nodi, vale la pena di vedere come avviene la scomposizione in capitoli e sezioni, dal momento che è qualcosa di più semplice, trattandosi di un concetto comune a molti altri sistemi di composizione. Semplificando le cose, si può affermare che un documento Texinfo è suddiviso in capitoli, che possono essere suddivisi

a loro volta in sezioni, sottosezioni e sotto-sottosezioni. Tuttavia, esistono diversi tipi di «capitoli» e di «sezioni»; la tabella u87.15 riepiloga questi comandi.

Tabella u87.15. Comandi per la suddivisione del testo in base al risultato stampato.

Comando	Descrizione
`@chapter`	Capitolo normale con numerazione.
`@section`	Sezione normale con numerazione.
`@subsection`	Sottosezione normale con numerazione.
`@subsubsection`	Sotto-sottosezione normale con numerazione.
`@unnumbered`	Capitolo senza numerazione.
`@unnumberedsec`	Sezione senza numerazione.
`@unnumberedsubsec`	Sottosezione senza numerazione.
`@unnumberedsubsubsec`	Sotto-sottosezione senza numerazione.
`@appendix`	Capitolo numerato in modo letterale -- appendice.
`@appendixsec`	Sezione di un'appendice.
`@appendixsubsec`	Sottosezione di un'appendice.
`@appendixsubsubsec`	Sotto-sottosezione di un'appendice.
`@majorheading`	Titolo importante senza numerazione e senza salto pagina.
`@chapterheading`	Capitolo senza numerazione e senza salto pagina.
`@heading`	Sezione senza numerazione.
`@subheading`	Sottosezione senza numerazione.
`@subsubheading`	Sotto-sottosezione senza numerazione.

In particolare, la suddivisione che fa capo al capitolo di tipo '**`@unnumbered`**', riguarda generalmente gli indici, o le introduzioni, mentre la suddivisione '**`@`**...**`heading`**', permette di realizzare dei documenti in forma di relazione.

L'esempio introduttivo mostra in che modo si definisce l'inizio di un capitolo, utilizzando il comando '**`@chapter`**' seguito dal titolo, senza bisogno che questo sia delimitato in qualche modo. La stessa cosa varrebbe per le sezioni e per le altre classificazioni inferiori.

```
@chapter Suddivisione del documento con Texinfo
```

Nodi

I nodi di Texinfo sono le unità di informazioni raggiungibili attraverso una navigazione ipertestuale. A differenza della struttura di capitoli e sezioni, i nodi sono unità non divisibili, quindi non esistono dei sotto-nodi. Questo fatto crea una sovrapposizione imperfetta tra la struttura a nodi e la struttura di capitoli e sezioni.

Il nodo viene dichiarato attraverso il comando '**`@node`**' e la sua estensione va da quel punto fino alla dichiarazione del nodo successivo. Convenzionalmente, si dichiarano i nodi subito prima di un capitolo, o di una sezione (o anche di una sottosezione, ecc.); in questo modo, la struttura a nodi ha una qualche corrispondenza con la struttura cartacea del documento. A questo proposito, è importante stabilire l'estensione dei nodi: per cominciare potrebbe essere conveniente avere nodi contenenti un capitolo intero; in questo caso si dichiarerebbero solo in corrispondenza di questi. Nel capitolo u88 viene trattato meglio il problema dell'abbinamento dei nodi con la struttura del documento, in particolare per gli automatismi che vengono offerti da Texinfo.

La struttura ipertestuale prevede un nodo di partenza obbligatorio, denominato '**`Top`**', all'interno del quale si trova normalmente un elenco di riferimenti, paragonabile a un indice generale, e una serie di altri nodi definiti dall'autore, con nomi liberi.


```
@node    Top, Suddivisione del documento, , (dir)
```

L'ipertesto Info, prevede l'aggregazione dei vari documenti scritti per questo sistema, attraverso un nodo precedente a quello '**`Top`**': si tratta del nodo '**`(dir)`**', corrispondente al file contenente l'indice iniziale di tutti i documenti Info installati effettivamente nel proprio sistema.

La dichiarazione di un nodo implica l'indicazione del suo nome, seguito da tre riferimenti ad altrettanti nodi: il nodo successivo, secondo un ordine ideale stabilito dall'autore; il nodo precedente; il nodo superiore.

```
@node nome_del_nodo, nodo_successivo, nodo_precedente, nodo_superiore
```

In pratica, questa struttura prevede una sequenza di nodi, stabilita in qualche modo, assieme al riferimento di un nodo di livello superiore. Il primo nodo in assoluto è '**`(dir)`**', mentre il primo nodo del documento è '**`Top`**'. Volendo utilizzare una struttura di nodi corrispondenti ai capitoli, senza suddivisioni ulteriori, si avrebbe una sola sequenza di nodi dal primo all'ultimo capitolo, dove per tutti l'unico nodo superiore sarebbe '**`Top`**'. Se invece si volesse realizzare una suddivisione maggiore, è ragionevole che i nodi contenuti in un capitolo formino una sequenza, dove il nodo superiore potrebbe essere quello iniziale del capitolo stesso.

Osservando la dichiarazione del nodo '**`Top`**' dell'esempio, si può vedere che il nodo precedente non è stato indicato, mentre il nodo superiore è '**`(dir)`**'. Infatti, non esiste un nodo precedente a '**`Top`**', mentre al di sopra di quello c'è solo l'indice generale, corrispondente al nome convenzionale '**`(dir)`**'.

Menù di riferimenti

La composizione in formato Info può generare automaticamente l'indice analitico, ma non l'indice generale. Per ottenere una sorta di indice generale, occorre indicare manualmente i riferimenti da qualche parte, di solito nel nodo '**`Top`**'. Nel caso dell'esempio introduttivo, sono stati indicati i riferimenti ai capitoli e all'indice analitico:

```
@menu
* Suddivisione del documento::      Il primo capitolo di questo esempio
                                    molto breve.
* Paragrafi::                       Il secondo capitolo.
* Indice analitico::                L'indice analitico.
@end menu
```

Come si vede, tra i comandi '**`@menu`**' e '**`@end menu`**', sono stati indicati i nomi dei nodi da raggiungere, seguiti da una descrizione. Le voci di questi menù possono essere più articolate, ma in generale, se possibile, conviene mantenere questa forma elementare:

```
* nome_nodo_da_raggiungere:: descrizione
```

Chi scrive un documento in Texinfo può anche fare a meno di preoccuparsi di questi menù, ma dovrebbe inserire almeno le voci che fanno riferimento ai nodi dell'indice analitico. Chi utilizza Emacs può anche ottenere la preparazione di questo menù automaticamente; per apprenderne il modo può consultare la documentazione originale su Texinfo.

Indici

Gli indici e i riferimenti di qualunque tipo siano, si ottengono attraverso l'indicazione di un'etichetta (da una parte) e di un riferimento che punta all'etichetta. Gli indici in particolare, sono una raccolta di riferimenti realizzata in modo automatico.

Tabella u87.19. Comandi per la gestione di indici generali o analitici.

Comando	Descrizione
`@contents`	Inserisce l'indice generale completo.
`@shortcontents`	Inserisce un indice generale ridotto.
`@summarycontents`	Inserisce un indice generale ridotto.
`@cindex` ***voce***	Inserisce una voce nell'indice analitico normale.
`@kindex` ***voce***	Inserisce una voce nell'indice dei comandi da tastiera.
`@pindex` ***voce***	Inserisce una voce nell'indice dei programmi.
`@findex` ***voce***	Inserisce una voce nell'indice delle funzioni.
`@vindex` ***voce***	Inserisce una voce nell'indice delle variabili.
`@tindex` ***voce***	Inserisce una voce nell'indice dei tipi di dati.
`@defindex` ***xy***	Crea l'indice ***xy***.
`@`***xy***`index` ***voce***	Inserisce una voce nell'indice ***xy***.
`@synindex` ***da a***	Trasferisce le voci di un indice in un altro.
`@syncodeindex` ***da a***	Come '**`@synindex`**' usando un carattere dattilografico.
`@printindex` ***xy***	Inserisce l'indice corrispondente alla sigla ***xy***.

Indice generale

L'indice generale viene realizzato solo nella composizione che si avvale di TeX, oltre che in quella per il formato HTML. Ciò avviene in modo automatico, indicando semplicemente il punto in cui questo deve apparire: l'inizio dei capitoli e delle classificazioni inferiori viene annotato nell'indice generale. Come accennato in precedenza, questo meccanismo non riguarda la composizione nel formato Info, per cui si utilizza un menù di riferimenti che fa le funzioni di indice generale.

L'indice generale può essere collocato all'inizio o alla fine del documento, in tal caso dopo gli indici analitici eventuali. Con il comando '**`@contents`**' si richiede la sua realizzazione in corrispondenza del comando stesso. È importante osservare che questo comando crea anche il titolo necessario, al contrario di ciò che avviene con l'indice analitico, come viene descritto tra poco.

Vale la pena di annotare il fatto che sono disponibili altri due comandi per ottenere la realizzazione di indici analitici meno dettagliati. Si tratta di '**`@shortcontents`**' e '**`@summarycontents`**'. La differenza tra i due sta solo nel titolo utilizzato per introdurli.

Indici analitici

Per quanto riguarda l'indice analitico, per ottenerlo è necessario indicare nel testo delle etichette apposite, con le quali si ottiene l'inserimento delle voci relative. Questo viene fatto più o meno come avviene in altri sistemi di composizione, ma con Texinfo occorre tenere conto di alcune particolarità che derivano dalla sua specializzazione ipertestuale. Prima di tutto, si deve considerare che secondo la politica di Texinfo, le etichette riferite a voci da inserire nell'indice analitico devono essere uniche. In questo modo si semplifica una serie di problemi nella navigazione di un documento Info, che non può essere ambigua. A questo proposito, la documentazione originale di Texinfo suggerisce di utilizzare voci descrittive piuttosto dettagliate. Nel momento in cui si devono usare voci descrittive, si può porre anche il problema del modo in cui il lettore va a cercarle nell'indice, per cui, se ci sono più modi per indicare lo stesso concetto, è meglio inserire più etichette alternative per l'indice analitico. Texinfo è in grado di gestire diversi indici analitici specifici:

- un indice normale per i concetti che vengono affrontati;

- un indice dei comandi da tastiera (combinazioni di tasti con funzionalità particolari nell'ambito di ciò che viene descritto);
- un indice dei programmi (nomi dei programmi eseguibili);
- un indice delle funzioni;
- un indice delle variabili;
- un indice dei tipi di dati.

I primi due tipi di indici dovrebbero avere un significato evidente; per gli altri, si fa riferimento a ciò che riguarda i linguaggi di programmazione.

Si intuisce che non è tecnicamente necessario utilizzare tutti questi indici. In generale, ci si potrebbe limitare all'inserimento delle voci nell'indice analitico normale. Tuttavia, se si vuole realizzare un documento in Texinfo, seguendo le convenzioni, è bene fare uso dell'indice giusto per ogni cosa. Questo permette in seguito l'aggregazione con altri documenti che hanno seguito le stesse convenzioni.

La sintassi per la dichiarazione di una voce nei vari indici analitici di Texinfo, può essere fatta secondo uno degli schemi seguenti, che rappresentano nell'ordine i tipi di indice descritti sopra:

```
@cindex voce_da_inserire_nell'indice_analitico_normale
```

```
@kindex voce_da_inserire_nell'indice_analitico_dei_comandi_da_tastiera
```

```
@pindex voce_da_inserire_nell'indice_analitico_dei_programmi
```

```
@findex voce_da_inserire_nell'indice_analitico_delle_funzioni
```

```
@vindex voce_da_inserire_nell'indice_analitico_delle_variabili
```

```
@tindex voce_da_inserire_nell'indice_analitico_dei_tipi_di_dati
```

A ogni tipo di indice è abbinata una sigla, il cui utilizzo viene descritto tra poco. La tabella u87.20 elenca queste sigle.

Tabella u87.20. Sigle dei vari tipi di indice analitico di Texinfo.

Sigla	Indice
`cp`	Indice analitico normale.
`ky`	Indice analitico dei comandi da tastiera.
`pg`	Indice analitico dei programmi.
`fn`	Indice analitico delle funzioni.
`vr`	Indice analitico delle variabili.
`tp`	Indice analitico dei tipi di dati.

Nell'esempio introduttivo è stato mostrato l'uso del comando '**`@cindex`**' per inserire alcune voci nell'indice analitico normale:

```
@chapter Suddivisione del documento con Texinfo
@cindex suddivisione
@cindex capitolo
@cindex sezione
@cindex sottosezione
```

La voce che si inserisce nell'indice analitico, può essere anche composta da più parole, separate normalmente da uno spazio, senza bisogno di utilizzare dei delimitatori di alcun tipo: l'interpretazione corretta del comando è garantita dal fatto che questo deve stare da solo in una riga.

L'inserimento nel documento finale di un indice analitico, viene ri-

chiesto in modo esplicito, attraverso il comando '**@printindex**', seguito dalla sigla corrispondente all'indice desiderato. In generale, questo viene fatto all'interno di un capitolo non numerato, come è stato mostrato nell'esempio introduttivo, dove si vede la richiesta di creazione di un indice generale normale.

```
@unnumbered Indice analitico

@printindex cp
```

Definizione di indici aggiuntivi

Dovrebbe essere ormai chiaro che ogni indice va usato per il suo scopo, altrimenti diventa impossibile la fusione di più documenti in un libro unico. Tuttavia, di fronte a questa filosofia di Texinfo che distingue tra gli indici, ci si può trovare di fronte all'esigenza di crearne degli altri. Questo si può fare semplicemente, attribuendo a questi indici particolari una sigla di due sole lettere.

Per fare un esempio, si potrebbe desiderare l'introduzione di un indice specifico per raccogliere gli elementi SGML di un DTD. Volendo chiamare questo indice con la sigla '**ml**', si dichiara l'utilizzo di un tale indice nell'intestazione del sorgente Texinfo con il comando '**defindex**':

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename esempio.info
@settitle Introduzione a Texinfo
@defindex ml
@c %**end of header
```

La sintassi di questo comando non prevede altro, per cui non viene mostrata formalmente. Per inserire una voce in un indice definito in questo modo, si usa il comando '***xy*index**', dove ***xy*** sono le due lettere che lo definiscono.

```
@chapter Gli elementi interni al testo
@mlindex em
@mlindex strong
```

L'esempio mostra l'inserimento delle voci '**em**' e '**strong**'.

Per ottenere l'inserimento dell'indice si procede come avviene già per gli indici già previsti:

```
@printindex ml
```

Fusione di indici

La suddivisione dettagliata delle voci da inserire nell'indice analitico è una cosa ragionevole solo in quanto resta possibile la fusione di più gruppi assieme, in un indice unico. Ciò si ottiene con i comandi '**@synindex**' e '**@syncodeindex**' che hanno la stessa sintassi:

```
@synindex sigla_indice_di_origine sigla_indice_di_destinazione
```

```
@syncodeindex sigla_indice_di_origine sigla_indice_di_destinazione
```

Questi comandi si possono inserire solo all'inizio del documento, preferibilmente nell'intestazione, come si vede nell'esempio seguente:

```
@c %**start of header
@setfilename sgmltexi.info
@settitle Sgmltexi
...
@syncodeindex ml cp
@syncodeindex vr cp
@c %**end of header
```

In questo caso, si fa in modo di riversare le voci dell'indice '**ml**' e '**vr**' nell'indice standard ('**cp**').

I due comandi si distinguono perché nel primo caso le voci vengono trasferite in modo normale, mentre nel secondo queste vengono trasformate per renderle nella destinazione in modo dattilografico (praticamente avvolte nel comando '**@code{}**').

Aspetto del testo e ambienti speciali

Texinfo prevede una serie di comandi per delimitare parti di testo riferite a oggetti particolari; nello stesso modo, ci sono altri comandi per definire delle forme di enfatizzazione, senza stabilire le caratteristiche di ciò che si indica. Le tabelle u87.27 e u87.28 elencano questi comandi, quando sono riferiti a parole e frasi inserite nel corpo normale.

Tabella u87.27. Comandi per delimitare oggetti particolari nel testo.

Comando	Descrizione
`@code{`***testo***`}`	Esempio letterale di un pezzo di programma.
`@kbd{`***testo***`}`	Digitazione dalla tastiera.
`@key{`***testo***`}`	Nome convenzionale di un tasto.
`@samp{`***testo***`}`	Esempio letterale di una sequenza di caratteri.
`@var{`***testo***`}`	Variabile metasintattica.
`@url{`***testo***`}`	Indirizzo URI.
`@file{`***testo***`}`	Nome di un file.
`@email{`***testo***`}`	Indirizzo di posta elettronica.
`@dfn{`***testo***`}`	Una definizione introdotta per la prima volta.
`@cite{`***testo***`}`	Riferimento bibliografico (titolo di un libro).

Tabella u87.28. Comandi per l'enfatizzazione generica.

Comando	Descrizione
`@emph{`***testo***`}`	Enfatizzazione normale.
`@strong{`***testo***`}`	Enfatizzazione più evidente.
`@sc{`***testo***`}`	Maiuscoletto.
`@r{`***testo***`}`	Carattere tondo normale.

La tabella u87.30 mostra l'elenco dei comandi riferiti ad ambienti particolari, usati per mostrare esempi, per le citazioni, oltre che per altre forme di evidenziamento del testo. I comandi in questione si usano da soli su una riga; hanno un'apertura e una chiusura, secondo la forma '***@comando***' e '**@end** ***comando***'. Per esempio, nel caso della citazione da mettere in evidenza, si può fare come nell'esempio seguente:

```
@quotation
Questa è una citazione.
@end quotation
```

Tabella u87.30. Comandi per delimitare blocchi di testo con funzioni specifiche.

Comando	Descrizione
`@quotation`	Testo citato.
`@example`	Codice, comandi e simili.
`@smallexample`	Come '**@example**', ma più piccolo.
`@lisp`	Codice LISP.
`@smalllisp`	Come '**@lisp**', ma più piccolo.
`@display`	Testo illustrativo senza uno scopo specifico.
`@smalldisplay`	Come '**@display**', ma più piccolo.
`@format`	Testo illustrativo, rispettando le interruzioni di riga.
`@smallformat`	Come '**@format**', ma più piccolo.
`@cartouche`	Disegna un riquadro arrotondato attorno al testo.

Anche se ciò non riguarda precisamente l'argomento di questa sezione, vale la pena di mostrare brevemente come si dichiara una nota a

piè pagina:

```
@footnote{testo}
```

Elenchi e tabelle

Gli elenchi vengono realizzati in Texinfo, più o meno come avviene con altri linguaggi di composizione. Dal momento che si deve poter arrivare al formato Info, le tabelle che vengono gestite sono molto simili a degli elenchi, pertanto è corretto trattare i due argomenti assieme.

Elenco puntato e numerato

L'elenco puntato viene delimitato dai comandi '**@itemize**' e '**@end itemize**'. Gli elementi dell'elenco vengono introdotti dal comando '**@item**'. Il simbolo usato per segnalare l'inizio dei vari elementi dell'elenco, viene dichiarato esplicitamente attraverso un argomento del comando '**@itemize**', '**@bullet**' o '**@minus**', che si riferiscono rispettivamente a un pallino (o un asterisco) e a un trattino. L'esempio seguente mostra un elenco con due voci principali, dove la prima si scompone in altre due voci inferiori; le voci principali sono introdotte da un pallino, mentre quelle inferiori sono evidenziate da un trattino.

```
@itemize @bullet
@item
primo elemento principale;

@itemize @minus
@item
primo sottoelemento;

@item
secondo sottoelemento;
@end itemize

@item
secondo elemento principale.
@end itemize
```

Gli spazi tra le voci sono opportuni, per ottenere un buon risultato nella composizione in formato Info, mentre per la composizione attraverso TeX, la cosa è indifferente.

L'elenco numerato è simile a quello puntato e si distingue solo perché è racchiuso tra i comandi '**@enumerate**' e '**@end enumerate**'. In particolare, in questo caso, al posto di definire il tipo di pallino da utilizzare, è possibile specificare il primo valore da utilizzare nell'elenco: se si tratta di un numero, quello diviene il primo valore di un elenco numerato vero e proprio; se si tratta di una lettera, si ottiene di un elenco numerato in modo letterale, a partire da quella lettera. L'esempio seguente mostra un elenco numerato che parte dal numero zero (quando di solito partirebbe da uno).

```
@enumerate 0
@item
primo elemento;

@item
secondo elemento;

@item
terzo elemento.
@end enumerate
```

Elenco descrittivo

L'elenco descrittivo è quello che per ogni punto mostra una parola o una frase a cui associa una descrizione. Per Texinfo, gli elenchi descrittivi sono delle tabelle a due colonne. L'ambiente viene delimitato dai comandi '**@table**' e '**@end table**', dove in particolare, il primo riceve un argomento che specifica il tipo di composizione da utilizzare per la prima colonna di questa specie di tabella. Per esempio,

```
@table @file
@item /etc/passwd
il file degli utenti;

@item /etc/group
il file dei gruppi.
@end @table
```


fa in modo che «/etc/passwd» e «/etc/group» vengano delimitati automaticamente con il comando '**@file**', mentre il resto, cioè la loro descrizione, viene lasciata con il carattere normale del testo.

Si può osservare che in questo caso il comando '**@item**' ha un argomento, che rappresenta la voce descrittiva dell'elenco. In situazioni particolari, può essere necessario indicare due voci assieme per la stessa descrizione; per questo esiste il comando '**@itemx**' che può essere usato subito dopo un comando '**@item**' normale.

```
@table @file
@item /etc/passwd
@itemx /etc/group
i file degli utenti e dei gruppi;

@item /etc/printcap
il file di configurazione del sistema di stampa.
@end @table
```

Esistono due varianti al comando '**@table**': si tratta di '**@ftable**' e '**@vtable**'. Il loro funzionamento è identico a '**@table**', con l'unica aggiunta che le voci indicate come argomento dei comandi '**@item**' o '**itemx**' vengono inserite automaticamente nell'indice delle funzioni e delle variabili, rispettivamente.

```
@vtable @code
@item PATH
contiene l'elenco dei percorsi per i file eseguibili;

@item HOME
contiene l'indicazione della directory personale abbinata
all'utente attuale.
@end @vtable
```

Tabelle vere e proprie

Le tabelle vere e proprie di Texinfo sono delimitate attraverso i comandi '**@multitable**' e '**@end multitable**'. Il comando di apertura richiede l'indicazione di altre informazioni che permettono di determinare l'ampiezza delle varie colonne. A questo proposito può essere usato il comando '**@columnfractions**', oppure degli esempi di testo:

```
@multitable @columnfractions frazione…
…
@end multitable
```

```
@multitable {testo_di_esempio}…
…
@end multitable
```

Il testo di esempio va racchiuso tra parentesi graffe, che quindi fanno parte del comando. Le frazioni sono valori decimali, la cui somma complessiva dovrebbe dare l'unità o un valore inferiore.

```
@multitable @columnfractions .2 .3 .5
<synellipsis>
@end multitable
```

L'esempio mostra il caso si una tabella che prevede tre colonne, dove la prima occupa un'ampiezza pari al 20 % del totale, la seconda il 30 % e l'ultima il restante 50 %. Se non si desiderano indicare questi valori percentuali si può usare l'altro metodo, come nell'esempio seguente:

```
@multitable {bla bla} {bla bla bla} {bla bla bla bla bla}
…
@end multitable
```

Le righe di queste tabelle sono introdotte dal comando '**@item**', a cui segue il testo della prima colonna. Il testo delle colonne successive viene introdotto da uno o più comandi '**@tab**'. L'esempio seguente mostra una tabella con tre colonne:

```
@multitable @columnfractions .25 .25 .5
@item colore
@tab valore
@tab annotazioni
@item nero
@tab 0
@tab il colore iniziale
@item marrone
@tab 1
@tab
@item rosso
@tab 2
```

```
@tab
…
@item bianco
@tab 9
@tab il colore finale
@end multitable
```

Modifica dello stile TeX

Lo stile standard predisposto per la composizione attraverso TeX potrebbe richiedere delle modifiche per qualche ragione. In generale, non è il caso di modificare il file che rappresenta lo stile standard, dal momento che è sufficiente farsene una copia da tenere assieme al sorgente Texinfo: quando si procede alla composizione, il file di stile '`texinfo.tex`' che si trova nella directory corrente, ha la precedenza.

Nell'estratto seguente vengono mostrate le righe utili che possono essere modificate per ottenere una traduzione dei termini che vengono inseriti automaticamente:

```
% Definizioni in italiano.
\ifx\putwordAppendix\undefined \gdef\putwordAppendix{Appendice}\fi
\ifx\putwordChapter\undefined  \gdef\putwordChapter{Capitolo}\fi
\ifx\putwordfile\undefined     \gdef\putwordfile{file}\fi
\ifx\putwordInfo\undefined     \gdef\putwordInfo{Info}\fi
\ifx\putwordMethodon\undefined \gdef\putwordMethodon{Method on}\fi
\ifx\putwordon\undefined       \gdef\putwordon{on}\fi
\ifx\putwordpage\undefined     \gdef\putwordpage{pagina}\fi
\ifx\putwordsection\undefined  \gdef\putwordsection{sezione}\fi
\ifx\putwordSection\undefined  \gdef\putwordSection{Sezione}\fi
\ifx\putwordsee\undefined      \gdef\putwordsee{vedere}\fi
\ifx\putwordSee\undefined      \gdef\putwordSee{Vedere}\fi
\ifx\putwordShortContents\undefined  \gdef\putwordShortContents{Indice breve}\fi
\ifx\putwordTableofContents\undefined\gdef\putwordTableofContents{Indice generale}\fi
```

Localizzazione

Il lavoro di nazionalizzazione del sistema Texinfo è ancora in corso. Recentemente è stato introdotto il comando '**`@documentlanguage`**' con il quale si ottiene la conversione automatica dei termini che vengono inseriti automaticamente in fase di composizione. Il comando riceve un argomento corrispondente alla sigla della lingua a cui si vuole fare riferimento, espressa secondo lo standard ISO 639 (sezione 13.3). Il comando si colloca preferibilmente nell'intestazione del sorgente. L'esempio seguente mostra la selezione della lingua italiana.

```
@c %**start of header
@setfilename prova.info
@settitle Prova
…
@documentlanguage it
@c %**end of header
```

Riferimenti

- Robert J. Chassel, Richard Stallman, *GNU Texinfo*, Free Software Foundation, Inc.
  http://www.gnu.org/software/texinfo/manual/texinfo/

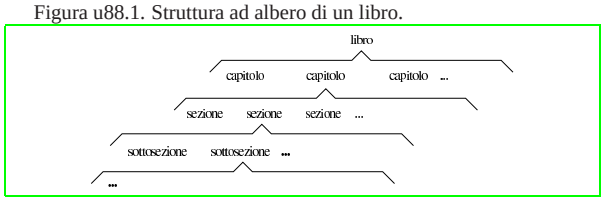
Texinfo: libro e ipertesto

Sequenza dei nodi secondo Texinfo 618
Definizione automatica della sequenza dei nodi e problemi relativi 619
Limitazioni originali della struttura a nodi 620
Riferimenti ipertestuali e limitazioni verbali 620
Altri tipi di riferimento 622
Riepilogo dei comandi relativi a nodi, ancore e riferimenti ... 623

Nel capitolo introduttivo è già stato affrontato il problema della gestione dei nodi in un documento Texinfo, ma alcuni aspetti sono stati solo sfiorati. Texinfo non può essere considerato pensando esclusivamente a uno dei risultati di composizione finale che possono essere generati, altrimenti si perde di vista la logica complessiva. In generale si sovrappongono due esigenze: il documento cartaceo da sfogliare e il documento elettronico da attraversare in modo ipertestuale.

Il documento cartaceo, ovvero il libro, ha una struttura ad albero che deriva dalla tradizione. Semplificando molto le cose si può rappresentare come nella figura u88.1, dove si vede che tutto viene suddiviso in capitoli, che possono eventualmente essere suddivisi ulteriormente in segmenti di livello inferiore (le sezioni).

Figura u88.1. Struttura ad albero di un libro.



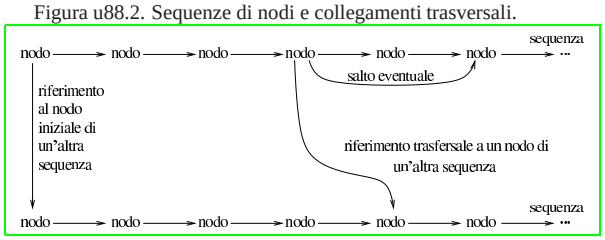
Un libro particolarmente corposo potrebbe anche raggruppare assieme i capitoli in parti; mentre un'opera potrebbe anche essere suddivisa in raggruppamenti ancora più grandi, che di solito corrispondono ai volumi, ovvero ai tomi.

La suddivisione ad albero mostrata nella figura, non basta a descrivere la struttura di un libro. Infatti, occorre considerare che i capitoli, se suddivisi in sezioni, non sono composti semplicemente dalla somma di queste sezioni, in quanto, prima di tali suddivisioni introducono il problema, che poi viene descritto in modo particolareggiato. In pratica, è come se ogni capitolo suddiviso in sezioni contenesse una sezione fantasma iniziale. Lo stesso ragionamento vale per le sezioni che si articolano in sottosezioni e così via con le classificazioni inferiori.

Lo stesso discorso può valere per la classificazione in parti e in tomi, anche se in questi casi, le informazioni che precedono i capitoli, o le parti, tendono a non avere la stessa valenza.

Il documento elettronico ipertestuale è composto da blocchi di informazioni che Texinfo definisce opportunamente come nodi. Generalmente questi nodi sono indivisibili, come avviene con Texinfo, ma tendono a essere raggruppati in sequenze ideali, oltre che prevedere la possibilità di saltare ad altri nodi (e quindi ad altre sequenze potenziali) attraverso riferimenti trasversali.

Figura u88.2. Sequenze di nodi e collegamenti trasversali.



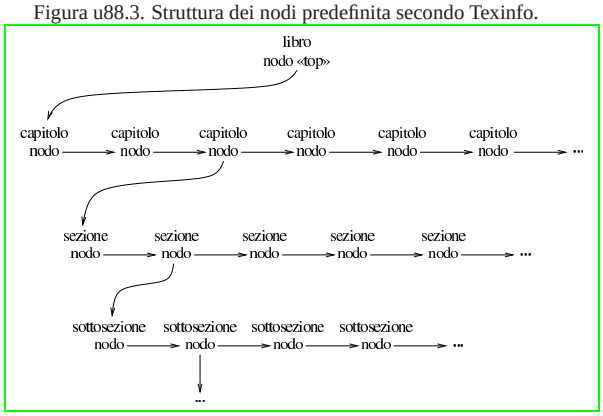
L'ipertesto puro è un documento in cui le informazioni sono raggiunte con un ordine che viene deciso durante la lettura, dove non c'è

l'esigenza di leggere tutto e non ci si preoccupa se volontariamente o inavvertitamente si salta qualche nodo che compone l'ipertesto complessivo. In generale è proprio questo il problema: un ipertesto non dispone necessariamente di un percorso predefinito di lettura.

Nel momento in cui si intende realizzare un documento unico, simultaneamente libro e ipertesto, si deve giungere in qualche modo a un compromesso. Texinfo consente di realizzare un ipertesto estremamente complesso, oppure un libro tradizionale; se però si vogliono fare entrambe le cose, di solito conviene realizzare l'ipertesto secondo la struttura stessa del libro.

Texinfo propone una sequenza predefinita, che viene generata automaticamente quando i nodi vengono dichiarati subito prima delle suddivisioni tradizionali del documento (capitoli, sezioni, ecc.), senza specificare la sequenza a cui appartengono. In generale si formano delle sequenze gerarchiche di questi nodi, dove si può passare ai livelli inferiori solo attraverso dei salti aggiuntivi, come si vede nella figura u88.3.

Figura u88.3. Struttura dei nodi predefinita secondo Texinfo.



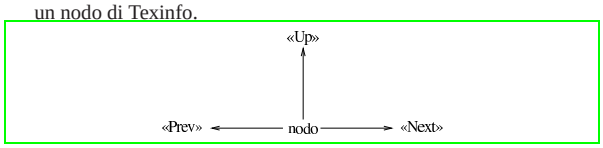
I salti aggiuntivi che permettono di raggiungere una sequenza inferiore di nodi vengono raggruppati convenzionalmente in un menù finale di riferimenti.

Sequenza dei nodi secondo Texinfo

Secondo Texinfo, i nodi hanno tre riferimenti che servono a comporre le sequenze e a legare tali sequenze in una dipendenza gerarchica. Si tratta di:

- '**Next**' il riferimento al nodo successivo nella sequenza;
- '**Prev**' il riferimento al nodo precedente nella sequenza;
- '**Up**' il riferimento al nodo gerarchicamente precedente.

Figura u88.4. Riferimenti standard che si diramano a partire da un nodo di Texinfo.



Il primo nodo di un documento Texinfo deve essere denominato '**Top**', che corrisponde idealmente a tutto ciò che precede il contenuto vero e proprio di un libro (subito dopo la copertina fino alla prefazione esclusa).

Per riprodurre lo schema gerarchico predefinito a cui si accennava in precedenza, dove si distinguono delle sequenze di nodi distinte a livelli diversi, è necessario inserire alla fine del testo di questi nodi un menù di riferimenti ai nodi di una sequenza inferiore. In generale, il nodo '**Top**' prevede l'inserimento di un menù di riferimenti ai nodi della sequenza principale, corrispondente in pratica ai capitoli di un libro; se i capitoli si articolano in strutture inferiori, anche i nodi relativi devono disporre di un menù che faccia riferimento alle sezioni, continuando così fino all'ultimo livello che si deve raggiungere.

Tuttavia, resta il fatto che tutto questo non sia indispensabile, dal momento che le sequenze dei nodi potrebbero essere determinate in modo arbitrario, senza rispettare la struttura tipica di un libro; senza avere così la necessità di definire tutti questi menù.

È proprio questa idea legata alla presenza di sequenze di nodi separate gerarchicamente che impone la presenza dei menù e di conseguenza impone l'esistenza del nodo '**Top**'. A questo punto, è il caso di osservare che il nodo '**Top**' non può appartenere a una sequenza di altri nodi allo stesso suo livello; per questo, viene inserito normalmente nella stessa sequenza dei capitoli.

Definizione automatica della sequenza dei nodi e problemi relativi

Texinfo prevede una struttura predefinita dei nodi che lo compongono, compatibile con la struttura di un libro, nel modo che è già stato mostrato in precedenza (figura u88.3). Per raggiungere questo risultato, nel sorgente Texinfo si indicano i nodi specificando solo il nome (senza stabilire relazioni con il nodo precedente, quello successivo e quello superiore). Tuttavia, questo non basta, perché se dal nodo si deve articolare una sequenza inferiore gerarchicamente, è necessario predisporre anche il menù relativo. L'esempio seguente rappresenta un nodo corrispondente a un capitolo, estratto dallo stesso sorgente dalla documentazione di Texinfo:

```
@node Overview
@chapter Overview of Texinfo

@dfn{Texinfo} is a documentation system that uses a single source file
to produce both online information and printed output.  This means that
instead of writing two different documents, one for the online
information and the other for a printed work, you need write only one
document.  Therefore, when the work is revised, you need revise only
that one document.

@menu
* Reporting Bugs::                Submitting effective bug reports.
* Using Texinfo::                 Create printed or online output.
* Info Files::                    What is an Info file?
* Printed Books::                 Characteristics of a printed book or manual.
* Formatting Commands::           @@-commands are used for formatting.
* Conventions::                   General rules for writing a Texinfo file.
* Comments::                      Writing comments and ignored text in general.
* Minimum::                       What a Texinfo file must have.
* Six Parts::                     Usually, a Texinfo file has six parts.
* Short Sample::                  A short sample Texinfo file.
* Acknowledgements and History::  Contributors and genesis.
@end menu
```

Il capitolo si articola in diverse sezioni e tutte devono essere elencate nel menù che si vede. Questo fatto può essere sentito come una limitazione, che bene o male costringe l'autore a curarsi della realizzazione di questi riferimenti ipertestuali. Oltre a questo, è il caso di considerare il modo in cui si presenta il documento quando viene fatta la composizione in forma Info: quando si accede al nodo del capitolo, si vedono solo quelle poche righe iniziali, mentre per entrare nelle sezioni successive occorre passare per la selezione del menù.

Il problema della predisposizione di questi menù si può risolvere utilizzando Emacs, attraverso alcuni comandi specifici della modalità Texinfo. Tuttavia, questa non è la soluzione definitiva, dal momento che si costringe a utilizzare Emacs, mentre chi non vuole farlo resta costretto ad arrangiarsi a mano.

Il problema dei capitoli spezzati in nodi separati è più serio. In effetti la suddivisione fatta attraverso le sequenze gerarchiche di nodi è perfettamente logica; tuttavia, nel momento in cui ci si accinge a leggere un documento del genere, sarebbe forse più logico scorrere il capitolo verticalmente per raggiungere le sue classificazioni inferiori come se si trattasse di un'unica pagina. Purtroppo, il sistema Info non consente di avere dei sottonodi, ovvero dei riferimenti a posizioni intermedie di un nodo, come invece avviene con l'HTML. Per risolvere in pratica questa limitazione bisogna limitarsi ad attribuire i nodi ai capitoli, tenendo presente che in questo modo non è possibile indicare nel testo dei riferimenti diretti a classificazioni inferiori.

Tuttavia, recentemente è stato introdotto il comando '**@anchor**' con cui si ottiene l'inserimento di un'etichetta raggiungibile come se si trattasse di un nodo, smussando così il problema dei nodi.

Limitazioni originali della struttura a nodi

Originariamente i nodi di Texinfo rappresentavano gli unici oggetti che potevano essere raggiunti attraverso riferimenti ipertestuali. In pratica, per fare riferimento a un capitolo o a una sezione, occorreva definire il nodo relativo per poi poter utilizzare comandi della serie '**@…ref**'.

È già stato visto che in generale conviene definire dei nodi in corrispondenza di tutti i capitoli, in modo da creare una sequenza definita a partire dal menù collocato nel nodo '**Top**'. Tuttavia, se c'è la necessità di fare riferimento a una sezione particolare di un certo capitolo, diventerebbe necessario dichiarare anche lì un nodo. Ma non basta definire un nodo in una sezione lasciando stare le altre sezioni, perché si creerebbe disordine nell'insieme; in pratica, se si definisce un nodo per una sezione di un certo capitolo, diventa indispensabile definire i nodi per le altre sezioni della stesso capitolo, avendo poi cura di predisporre il menù necessario.

A questo problema si è posto rimedio aggiungendo il comando '**@ancor{}**' che ha lo scopo di collocare un'ancora, ovvero un'etichetta raggiungibile attraverso riferimenti ipertestuali, senza dichiarare implicitamente l'inizio di un nodo in quella posizione.

```
@ancor{nome_ancora}
```

I nomi delle ancore appartengono allo stesso dominio dei nomi dei nodi, per cui non si devono creare dei conflitti nella scelta dei nomi. Inoltre, quando si fa riferimento a un'ancora, nella composizione Info si ottiene normalmente di raggiungere l'inizio del nodo in cui si trova.

Anche la gestione degli indici analitici è condizionata dalla struttura a nodi di Texinfo. In generale non è indispensabile che la voce da collocare in un indice analitico si trovi all'inizio di un nodo; quando però dall'indice analitico si vuole raggiungere il testo in cui questa è stata dichiarata, si arriva in realtà all'inizio del nodo in cui questa si trova. Se la voce si trova all'interno di una piccola sottosezione, mentre l'unico nodo disponibile è quello che fa capo al capitolo, si raggiunge l'inizio del capitolo. Naturalmente, questo vale per la navigazione di un documento che è stato composto in formato Info, mentre nelle altre forme di composizione il problema scompare o viene attenuato.

Riferimenti ipertestuali e limitazioni verbali

Texinfo nasce come un sistema di composizione per documentazione scritta in lingua inglese. Attualmente il lavoro attorno a Texinfo si rivolge anche verso le esigenze delle altre lingue, selezionabili attraverso il comando '**@documentlanguage**', ma questo lavoro non è ancora completo nel momento in cui si scrivono queste note.

Texinfo dispone di quattro comandi diversi per i riferimenti ipertestuali, il cui scopo è quello di adattarsi alle esigenze del contesto. Ma in questo caso, il contesto è prevalentemente di tipo «verbale». Vale la pena di descrivere brevemente questi quattro comandi, mostrando le conseguenze pratiche del loro utilizzo. Qui non vengono mostrate tutte le varianti perché ciò richiederebbe un capitolo apposito, mentre la documentazione originale è molto chiara a questo proposito.[1]

Vengono considerati i comportamenti confrontando solo la composizione Info e quella stampata (DVI, PostScript e PDF), perché l'HTML non ha ancora una sistemazione definitiva.

Questi comandi ricevono più argomenti distinti in base all'uso di una virgola di separazione. Per questa ragione la virgola non può essere usata all'interno di un argomento. Si tratta evidentemente di una limitazione importante da tenere in considerazione.

```
@xref{nodo, titolo_per_info, titolo_o_argomento, file_info, titolo_del_documento_stan
```

Il comando '**@xref{}**' consente di ottenere dei riferimenti ipertestuali molto descrittivi. In generale è obbligatoria l'indicazione del nodo (il primo argomento), mentre il resto può essere omesso. Dopo l'indicazione del nodo, alcuni argomenti successivi riguardano esclusivamente la composizione Info, mentre gli altri solo la composizione stampata (e simili). Nella composizione Info viene indicato il nome del nodo; se fornito appare il titolo specifico per la composizione Info ed eventualmente anche il nome del file Info esterno in cui cercarlo. Nella composizione per la stampa si ha l'indicazione del titolo dell'argomento e se non viene fornita questa indicazione ci si limita a mostrare il nome del nodo stesso; se poi il riferimento è interno al documento viene aggiunta l'indicazione della pagina, altrimenti diventa necessario fornire il titolo del documento esterno che così appare al posto del numero della pagina.

L'uso più semplice di '**@xref{}**' è quello in cui si indica solo il nodo, come nell'esempio seguente:

```
Bla bla bla. @xref{Din don dan}. Bla bla bla...
```

Il risultato nell'ipertesto Info è:

```
Bla bla bla. *Note Din don dan::. Bla bla bla...
```

mentre con la composizione per la stampa l'aspetto è molto diverso:

```
Bla bla bla. See Chapter 3 [Din don dan], page 22. Bla bla bla...
```

Tanto per cominciare si può comprendere che si tratta di un riferimento che può essere collocato solo all'inizio di una frase (di un testo inglese), dal momento che la prima parola, '**See**', ha l'iniziale maiuscola. Eventualmente non è detto che il riferimento debba concludersi con un punto fermo come avviene nell'esempio, ma la frase che continua è comunque condizionata dal modo in cui viene rappresentato tale riferimento.

```
@ref{nodo, titolo_per_info, titolo_o_argomento, file_info, titolo_del_documento_stamp
```

Il comando '**@ref{}**' si comporta in modo analogo a '**@xref{}**' con la differenza che nella composizione per la stampa non viene generata la parola '**See**' iniziale. Ciò consente di collocare il riferimento alla fine di una frase, oppure, con l'accortezza necessaria, anche in mezzo.

```
@pxref{nodo, titolo_per_info, titolo_o_argomento, file_info, titolo_del_documento_sta
```

Il comando '**@pxref{}**' è il più strano per chi non scrive utilizzando la lingua inglese. La lettera «p» sta per «parentheses», cioè parentesi, quelle all'interno delle quali dovrebbe essere collocato. A differenza del comando '**@xref{}**', la composizione per la stampa inizia con '**see**' (iniziale minuscola), mentre la composizione Info aggiunge un punto fermo.

```
@inforef{nodo, titolo, file_info}
```

Il comando '**@inforef{}**' serve a fare riferimento a un file Info esterno, per il quale non si vuole o non si può fare riferimento a un'analoga versione stampata. Mentre nella composizione Info il risultato è uguale a quanto è già stato visto per '**@xref{}**', nella composizione stampata viene fatto esplicito riferimento a un file Info. Si può ottenere una cosa simile a quella seguente:

```
Bla bla bla. See Info file 'miofile', node 'Din don dan'. Bla bla bla...
```

Si intende che il riferimento sia fatto per essere collocato esattamente all'inizio di un periodo, dato il fatto che anche qui la parola '**See**' ha l'iniziale maiuscola.

Tabella u88.10. Comandi per i riferimenti incrociati.

Comando	Descrizione o risultato
`@anchor{`*etichetta*`}`	Inserisce un ancora nel testo.
`@xref`	«See...»
`@ref`	Come '**`@xref`**', ma senza «See».
`@pxref`	«see...»
`@inforef`	«See Info file...».

Altri tipi di riferimento

Texinfo dispone di altri tipi di riferimento che però risultano indolori dal punto di vista della lingua utilizzata per scrivere il proprio documento.

`@uref{`*uri*`,` *descrizione*`,` *testo_sostitutivo_dell'indirizzo*`}`

Il comando '**`@uref{}`**' consente di annotare un indirizzo URI secondo modalità differenti: se si indica solo il primo argomento, viene mostrato in ogni tipo di composizione; se appare anche il secondo argomento, vengono mostrate entrambe le cose, la descrizione e l'indirizzo, tranne nel caso della composizione HTML, in cui l'indirizzo non viene più mostrato; se si indica il terzo argomento (il secondo diventa superfluo), non si vuole mostrare l'indirizzo URI, mentre nella composizione HTML viene comunque attivato il riferimento. Si osservino gli esempi seguenti.

```
Bla bla bla @uref{http://www.dinkel.brot.dg/} bla bla bla...
```

Questo genera l'inserimento dell'indirizzo nel testo senza delimitazioni, in ogni tipo di composizione.

```
Bla bla bla @uref{http://www.dinkel.brot.dg/, Titolo} bla bla bla...
```

In questo modo, la composizione per la stampa e quella per Info generano un risultato del tipo:

```
Bla bla bla Titolo (http://www.dinkel.brot.dg/) bla bla bla...
```

Invece, nella composizione HTML l'indirizzo URI scompare dalla vista, nel modo seguente:

```
Bla bla bla <a href="http://www.dinkel.brot.dg/">Titolo</a> bla bla bla...
```

Infine, l'esempio seguente mostra l'uso del terzo argomento (si noti l'uso della coppia di virgole per segnalare l'assenza del secondo argomento):

```
Bla bla bla @uref{http://www.dinkel.brot.dg/,, Titolo} bla bla bla...
```

Nella composizione stampata e in quella Info si perde completamente l'informazione dell'indirizzo URI:

```
Bla bla bla Titolo bla bla bla...
```

Nella composizione HTML l'indirizzo URI rimane nascosto alla vista, come è già stato visto in precedenza:

```
Bla bla bla <a href="http://www.dinkel.brot.dg/">Titolo</a> bla bla bla...
```

A fianco di '**`@uref{}`**' si pone anche un comando specifico per l'annotazione di indirizzi di posta elettronica:

`@email{`*indirizzo*`,` *descrizione*`}`

Il comando '**`@email{}`**' si comporta in pratica come '**`@uref{}`**', con la differenza che il terzo argomento non esiste, per cui si mostra sempre l'indirizzo, che eventualmente viene preceduto dalla sua descrizione. Nel caso della composizione in HTML, viene generato un riferimento ipertestuale del tipo '**`mailto:`**'.

Esiste un altro modo di indicare un riferimento a un indirizzo URI. Si tratta del comando '**`@url{}`**', che serve solo a mostrare tale indirizzo, senza generare nel formato HTML alcun riferimento:

`@url{`*uri*`}`

L'indirizzo URI viene mostrato senza delimitazioni in ogni tipo di composizione. In generale può essere conveniente utilizzare questo comando al posto di '**`uref{}`**' quando si indica un indirizzo ipotetico o un indirizzo che non è più valido (al quale non sarebbe opportuno puntare con un riferimento ipertestuale).

Riepilogo dei comandi relativi a nodi, ancore e riferimenti

- `@node` *nome_del_nodo*`,` *nodo_successivo*`,` *nodo_precedente*`,` *nodo_superiore*

  Definizione di un nodo, da collocare subito prima di un capitolo, una sezione, o di un'altra classificazione analoga. Il comando occupa una riga.

- `@menu`
  `[`*testo_descrittivo*`]`
  voce_del_menù
  …
  `@end menu`

  Si tratta della definizione di un menù da collocare alla fine del testo di un nodo, per raggiungere una sequenza di nodi di livello inferiore. Le voci del menù possono avere due forme alternative:

  `* [(`*file_info*`)]`*nome_nodo*`::` *titolo_o_argomento*

  `*` *nome_della_voce*`: [(`*file_info*`)]`*nome_nodo*`.` *titolo_o_argomento*

  In generale, la seconda forma è usata molto poco.

- `@ancor{`*nome_ancora*`}`

  Definizione di un'ancora, ovvero un'etichetta a cui poter fare riferimento attraverso comandi '**`@…ref`**'. I nomi delle ancore e i nomi dei nodi appartengono allo stesso dominio.

- `@xref{`*nodo*`,` *titolo_per_info*`,` *titolo_o_argomento*`,` *file_info*`,` *titolo_del_documento_st*

  `@ref{`*nodo*`,` *titolo_per_info*`,` *titolo_o_argomento*`,` *file_info*`,` *titolo_del_documento_sta*

  `@pxref{`*nodo*`,` *titolo_per_info*`,` *titolo_o_argomento*`,` *file_info*`,` *titolo_del_documento_*

  Tre tipi complementari di riferimento a un nodo dello stesso documento o di un documento esterno, realizzato sempre con Texinfo. Nel primo caso il riferimento va posto all'inizio di un periodo; nel secondo può stare all'interno o alla fine di una frase; nel terzo caso deve essere collocato tra parentesi.

- `@inforef{`*nodo*`,` *titolo*`,` *file_info*`}`

  Riferimento a un nodo esterno di un documento disponibile solo in forma Info. Il riferimento va posto all'inizio di un periodo.

- `@uref{`*uri*`,` *descrizione*`,` *testo_sostitutivo_dell'indirizzo*`}`

  `@email{`*indirizzo*`,` *descrizione*`}`

Riferimento a un URI generico o a un indirizzo di posta elettronica, per il quale la composizione HTML genera un riferimento.

- `@url{`***uri***`}`

  Annotazione pura e semplice di un indirizzo URI senza creare alcun riferimento nella composizione HTML.

[1] È il caso di ricordare che le parentesi graffe fanno parte dei comandi di Texinfo.

Sgmltexi: installazione e utilizzo

Installazione di Sgmltexi 625
Gettext 625
Dipendenze 626
Come si usa il programma frontale 626
Riferimenti 628

Sgmltexi [1] è un DTD e un sistema frontale per la composizione in Texinfo a partire da un formato SGML. L'idea alla base di Sgmltexi è quella di avere la possibilità di scrivere un documento Texinfo attraverso la semplificazione e la guida che può dare un sistema SGML.

All'interno di Sgmltexi, la gestione dei nodi di Texinfo può avvenire in modo automatico e trasparente, generando un menù Info unico nel nodo '**`Top`**'. I nomi dei nodi, quando sono generati automaticamente, usano stringhe del tipo «cap 1», «app A»,...

Sgmltexi ha uno schema preciso: ci possono essere una o più introduzioni iniziali; nella parte centrale c'è un corpo che può essere scomposto in vario modo; ci possono essere delle appendici; al termine possono apparire degli indici analitici. Il corpo è organizzato in capitoli, che possono essere raggruppati in parti ed eventualmente anche in tomi. In tal modo, si possono gestire facilmente anche documenti di grandi dimensioni.

Sgmltexi è un progetto che non viene più sviluppato, in considerazione del fatto che la documentazione GNU tende a migrare verso Docbook, pur garantendo la compatibilità con Texinfo.

Installazione di Sgmltexi

Sgmltexi è composto da due eseguibili Perl: '**`sgmltexi`**' e '**`sgmltexi-sp2texi`**'. Questi due file devono essere collocati in una directory in cui possono essere avviati senza bisogno di indicare il percorso; in pratica in una directory elencata all'interno della variabile di ambiente '**`PATH`**'.

Evidentemente, è necessario l'interprete Perl; precisamente questi programmi cercano il file '`/usr/bin/perl`'. Se il proprio sistema operativo è organizzato diversamente, è necessario intervenire modificando la prima riga dei due eseguibili:

```
#!/usr/bin/perl
…
```

Sgmltexi si aspetta di trovare alcuni file:

- '`/etc/sgmltexi/sgmltexi.cat`'
  il catalogo SGML di Sgmltexi;
- '`/etc/sgmltexi/sgmltexi.dcl`'
  la dichiarazione SGML di Sgmltexi;
- '`/etc/sgmltexi/sgmltexi.dtd`'
  il DTD di Sgmltexi;
- '`/etc/sgmltexi/entities/`'
  la directory contenente i file delle entità SGML standard.

Tutti questi file dovrebbero trovarsi esattamente dove previsto; in alternativa si devono realizzare almeno dei collegamenti per ricreare i percorsi stabiliti.

Gettext

I messaggi di Sgmltexi possono essere tradotti. Per installare i file PO già esistenti è necessario compilarli come nell'esempio seguente:

```
$ msgfmt -vvvv -o sgmltexi.mo it.po [ Invio ]
```

In questo esempio, il file ‘`it.po`’ viene compilato generando il file ‘`sgmltexi.mo`’. Questo file può essere collocato in ‘`/usr/share/locale/it/LC_MESSAGES/`’, o in un’altra posizione analoga in base agli standard del proprio sistema operativo.

Se non è disponibile il modulo Perl-gettext, che serve a Sgmltexi per accedere alle traduzioni, è possibile eliminare il suo utilizzo e simulare la funzione di Gettext. In pratica si commentano le istruzioni seguenti:

```
# Non si vuole usare gettext.
#use POSIX;
#use Locale::gettext;
#setlocale (LC_MESSAGES, "");
#textdomain ("sgmltexi");
```

Inoltre, si tolgono i commenti dalla dichiarazione della funzione fittizia ‘**`gettext()`**’, come si vede qui:

```
sub gettext
{
    return $_[0];
}
```

Dipendenze

È il caso di riepilogare le dipendenze di Sgmltexi da altri applicativi:

- Perl

  dal momento che si tratta di un programma scritto in Perl, deve essere presente l’interprete relativo;

- SP o Jade

  per l’analisi SGML occorre il programma ‘**`nsgmls`**’ che fa parte del pacchetto SP o anche del pacchetto Jade;

- Perl-gettext

  per accedere ai messaggi tradotti del programma, è necessario il modulo Perl-gettext, salva la possibilità di escluderne l’utilizzo come è già stato mostrato;

- TeX e Texinfo

  per arrivare a una composizione finale è necessario ovviamente disporre di Texinfo, che potrebbe già essere integrato nella propria distribuzione TeX (di solito si tratta di teTeX).

Come si usa il programma frontale

Una volta preparato il sorgente in formato Sgmltexi, bisogna utilizzare il programma ‘**`sgmltexi`**’ per controllare l’elaborazione SGML e gli altri applicativi di composizione di Texinfo.

Di solito, la cosa migliore per iniziare lo studio di un sistema di composizione, è partire da un esempio banale, funzionante, che consenta di apprendere l’uso elementare degli strumenti relativi.

```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN">
<sgmltexi lang="it">
<head>
    <admin>
        <setfilename content="esempio.info">
        <settitle content="Esempio">
    </admin>
    <titlepage>
        <title>Esempio</title>
        <subtitle>Un esempio per un documento in formato Sgmltexi</subtitle>
        <abstract>
            <p>Questo è solo un esempio di un documento scritto
            utilizzando Sgmltexi.</p>
        </abstract>
        <author>Pinco Pallino &lt;ppinco@dinkel.brot.dg&gt;</author>
        <legal>
            <copyright>Copyright &copy; 2000 Pinco Pallino</copyright>
            <license>
                <p>Permission is granted to copy, distribute and/or
                modify this document under the terms of the GNU Free
                Documentation License, Version 1.1 or any later version
                published by the Free Software Foundation; with no
                Invariant Sections, with no Front-Cover Texts, and with
                no Back-Cover Texts. A copy of the license is included
                in the section entitled "GNU Free Documentation
                License".</p>
            </license>
        </legal>
    </titlepage>
    <contents>
</head>
<body>
<h1>Esempio generale</h1>
```



```
<p>Questo è l’esempio tipico di un capitolo di Sgmltexi...</p>

<p>Non c’è molto da scrivere in questo caso...</p>

</body>
</sgmltexi>
```

Supponendo di avere installato correttamente Sgmltexi (e anche Texinfo), supponendo inoltre che il file si chiami ‘`prova.sgml`’, si possono usare i comandi seguenti:

- `$` **`sgmltexi --sgml-check prova.sgml`** [*Invio*]

  per verificare la correttezza formale dell’SGML;

- `$` **`sgmltexi --texi prova.sgml`** [*Invio*]

  per ottenere semplicemente il file ‘`prova.texinfo`’, in formato Texinfo;

- `$` **`sgmltexi --info prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.info`’, in formato Info;

- `$` **`sgmltexi --dvi prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.dvi`’, in formato DVI;

- `$` **`sgmltexi --ps prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.ps`’, in formato PostScript;

- `$` **`sgmltexi --pdf prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.pdf`’, in formato PDF;

- `$` **`sgmltexi --html prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.html`’, in formato HTML;

- `$` **`sgmltexi --xml prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.xml`’, in formato XML di Texinfo;

- `$` **`sgmltexi --docbook prova.sgml`** [*Invio*]

  per ottenere il file ‘`prova.xml`’, in formato XML di Docbook.

La sintassi di ‘**`sgmltexi`**’ è quella che si vede nello schema seguente:

`sgmltexi` [***opzioni***] ***sorgente_sgml***

In generale, è bene che il nome del file sorgente in formato Sgmltexi abbia l’estensione standard ‘`.sgml`’.

Opzione	Descrizione
`--help`	Mostra una guida sintetica e termina di funzionare.
`--version`	Mostra le informazioni sulla versione e termina di funzionare.
`--force`	Quando il contesto lo consente, forza le situazioni. Può essere utile in particolare per la composizione in formato Info e in formato HTML, per passare la stessa opzione al programma ‘**`makeinfo`**’.
`--number-sections`	Numera le sezioni quando ciò non è previsto in modo normale.
`--clean`	Elimina i file intermedi che non servono, abbinati al nome del sorgente.
`--verbose`	Mostra più informazioni durante l’elaborazione.
`--input-encoding=`***codifica***	Stabilisce la codifica del file in ingresso, tenendo conto che sono ammissibili solo le parole chiave ‘**`ISO-8859-`*****n***’, dove ***n*** va da 1 a 10.
`--sgml-include=`***entità_parametrica*** `--include=`***entità_parametrica***	Assegna la parola chiave ‘**`INCLUDE`**’ all’entità parametrica SGML indicata. Questo serve ad abilitare l’inclusione di porzioni di sorgente SGML che sono controllate in questo modo.
`--paper=`***formato_composizione***	Serve a definire in qualche modo il formato finale stampato di composizione. Sono disponibili le parole chiave seguenti: ‘**`letter`**’, ‘**`a4`**’, ‘**`a4wide`**’, ‘**`a4latex`**’ e ‘**`small`**’.

Opzione	Descrizione
`--setchapternewpage={on|off|odd}`	Definisce l'inizio dei capitoli nella composizione per la stampa, ignorando il marcatore '**`<setchapternewpage content="…">`**' all'interno del sorgente del documento.
`--footnotestyle={end|separate}`	Definisce la collocazione delle note a piè pagina, ignorando il marcatore '**`<footnotestyle content="…">`**' all'interno del sorgente.
`--headings={on|off|single|double|singleafter|doubleafter}`	Attiva o disattiva le intestazioni, specificando eventualmente l'uso di intestazioni uguali o distinte. Questa opzione fa ignorare il marcatore '**`<headings content="…">`**' all'interno del sorgente del documento.
`--sgml-syntax` `--sgml-check`	Controlla la correttezza formale del sorgente SGML, mostrando gli errori trovati.
`--sp`	Genera un risultato «post-SP», nel senso che restituisce soltanto quanto ottenuto dall'analizzatore SGML, a scopo diagnostico.
`--texi` `--texinfo`	Genera un sorgente Texinfo.
`--dvi`	Compone generando un risultato in formato DVI.
`--ps` `--postscript`	Compone generando un formato in PostScript.
`--pdf`	Compone generando un formato PDF.
`--info`	Generate un risultato in formato Info.
`--text`	Genera un risultato in formato testo puro.
`--html`	Genera un risultato in formato HTML.
`--xml`	Genera un risultato in formato XML di Texinfo.
`--docbook`	Genera un risultato in formato XML Docbook.

- `$ sgmltexi --sgml-syntax prova.sgml` [*Invio*]

  Analizza la validità formale del sorgente '`prova.sgml`'.

- `$ sgmltexi --ps prova.sgml` [*Invio*]

  Genera un risultato in formato PostScript attraverso l'aiuto di '**`texi2dvi`**' e di '**`dvips`**'.

Riferimenti

- Daniele Giacomini, *Sgmltexi*
  extra/sgmltexi/
- Gaetano Paolone, *Sgmltexi*, pacchetto GNU/Linux Debian
  http://packages.debian.org/sgmltexi
  http://ftp.debian.org/debian/pool/main/s/sgmltexi/

[1] **Sgmltexi** GNU GPL

Sgmltexi: struttura

Struttura generale per un sorgente Sgmltexi 629
- Intestazione 632
- Informazioni amministrative 632
- Pagine iniziali 634
- Indice generale 636
- Nodi e menù Info iniziale 636
- Introduzione 637
- Corpo 637
- Appendice 638
- Indici analitici 638

Scomposizione del documento, nodi e menù Info 638
- Numerazione o meno dei titoli 639

Codifica 640
- Entità standard e non standard 640

Sgmltexi impone uno schema preciso al documento, in base alle consuetudini dei documenti stampati. Questo capitolo descrive brevemente tale struttura.

Struttura generale per un sorgente Sgmltexi

Il sorgente Sgmltexi tipico inizia così:

```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN">
```

Naturalmente, potrebbe essere conveniente la definizione iniziale di alcune entità generali, come si vede nell'esempio seguente:

```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN">
[
<!ENTITY EDITION   "2000.05.20">
...
...
]>
```

Tutto il documento viene racchiuso all'interno dell'elemento '**`sgmltexi`**', rispettando una certa struttura: deve esserci un elemento '**`head`**', ci può essere un elemento '**`intro`**', ci deve essere un elemento '**`body`**', infine ci può essere un elemento '**`appendix`**'. Lo spazio successivo all'elemento '**`appendix`**' può essere occupato da alcuni indici analitici (cosa che viene descritta meglio in seguito).

```
<sgmltexi>
<head>
...
</head>
<intro>
...
</intro>
<body>
...
</body>
<appendix>
...
</appendix>
</sgmltexi>
```

L'elemento '**`sgmltexi`**' ha tre attributi: '**`lang`**', '**`charset`**', '**`spacing`**'. Attraverso l'attributo '**`lang`**' si definisce il linguaggio in cui è scritto il documento, richiamando implicitamente una configurazione particolare all'interno di Texinfo. Questo linguaggio si indica assegnando una sigla corrispondente allo standard ISO 639 (sezione 13.3), come si vede nell'esempio seguente:

```
<sgmltexi lang="it">
```

L'attributo '**`charset`**' permette di indicare il valore da assegnare al comando '**`@documentencoding`**' di Texinfo. L'uso di questo attributo viene oscurato dall'opzione '**`--input-encoding`**', se questa viene usata. Infatti, tale opzione implica un'elaborazione del sorgente per cui si genera un file Texinfo in formato ISO 646 (ASCII tradizionale), cosa che fa perdere di significato al comando '**`@documentencoding`**'.

La composizione di un sorgente Texinfo dà risultati differenti a seconda dei casi, per cui alle volte può essere conveniente scrivere usando comandi come '**@'a**' («à»), mentre altre volte conviene scrivere usando una codifica ISO 8859-***n***, annotando questo nel comando '**@documentencoding**'. Probabilmente, è prevista la sistemazione di questo problema nelle prossime versioni di Texinfo; per ora l'ambivalenza di Sgmltexi può aiutare in tal senso.

L'attributo '**spacing**' dovrebbe essere superfluo, dal momento che serve a definire la spaziatura alla fine del punto fermo. Questo comportamento dovrebbe essere definito automaticamente in base alla scelta del linguaggio. Questo attributo consente quindi di forzare la situazione, imponendo una spaziatura non conforme allo standard. I valori che si possono assegnare sono: '**normal**', '**french**' e '**uniform**'. Assegnando '**french**', oppure '**uniform**', si ottiene in pratica la stessa cosa che si otterrebbe con il comando '**@frenchspacing**' di Texinfo. L'esempio seguente rappresenta ciò che potrebbe essere conveniente in un testo italiano:

```
<sgmltexi lang="it" charset="ISO-8859-1" spacing="uniform">
```

Tabella u90.6. Elementi SGML che compongono la struttura generale.

Elemento o attributo	Contenuto	Descrizione
sgmltexi		Contenitore del documento.
lang	Attributo	Sigla ISO 639 del linguaggio.
charset	Attributo	Codifica nella forma '**ISO-8859-*n***'.
spacing	Attributo	'**normal**', '**french**' e '**uniform**'.
head		Intestazione del documento.
admin		Informazioni amministrative.
setfilename	Vuoto	Inserisce il comando '**@setfilename**'.
content	Attributo	Il nome del primo file Info da generare.
settitle	Vuoto	Inserisce il comando '**@settitle**'.
content	Attributo	Titolo.
setchapternewpage	Vuoto	Inserisce il comando '**@setchapternewpage**'.
content	Attributo	Separazione dei capitoli: '**on**', '**off**', '**odd**'.
footnotestyle	Vuoto	Inserisce il comando '**@footnotestyle**'.
content	Attributo	Piè pagina: '**end**', '**separate**', '**empty**'.
headings	Vuoto	Inserisce il comando '**@headings**'.
content	Attributo	Intestazioni: '**on**', '**off**', '**single**', '**double**', '**singleafter**', '**doubleafter**'.
defindex	Vuoto	Inserisce il comando '**@defindex**'.
name	Attributo	Sigla di due lettere dell'indice analitico.
defcodeindex	Vuoto	Inserisce il comando '**@defcodeindex**'.
name	Attributo	Sigla di due lettere dell'indice analitico.
synindex	Vuoto	Inserisce il comando '**@synindex**'.
from	Attributo	L'indice di origine: una sigla di due lettere.
to	Attributo	L'indice di destinazione: una sigla di due lettere.
syncodeindex		Inserisce il comando '**@syncodeindex**'.
from	Attributo	L'indice di origine: una sigla di due lettere.

Elemento o attributo	Contenuto	Descrizione
to	Attributo	Destinazione in cui deve apparire in dattilografico.
infodir	Vuoto	Comando '**@direntry**' in modo automatico.
infodir	#PCDATA	Comando '**@direntry**' con un contenuto letterale.
titlepage		Informazioni delle prime pagine.
title	%inline;	Inserisce il comando '**@title**'.
subtitle	%inline;	Inserisce il comando '**@subtitle**'.
abstract	%block;	Descrizione del contenuto del documento.
author	%inline;	Inserisce il comando '**@author**'.
frontcovertext	%block;	Testo da inserire in copertina.
tpextra	%block;	Testo aggiuntivo nelle prime pagine.
legal		Informazioni legali alla base della seconda pagina.
copyright	%inline;	Una riga di copyright.
publishnote	%block;	Note da mostrare prima della licenza.
license	%block;	Condizioni con cui è rilasciato il documento.
coverart	%block;	Note sulla copertina, da mostrare dopo la licenza.
dedications	%block;	Pagina delle dediche.
contents	Vuoto	Indice generale standard.
shortcontents	Vuoto	Indice generale ridotto.
summarycontents	Vuoto	Indice generale ridotto.
menu	Vuoto	Inserisce un menù Info automatico.
topnode	Vuoto	Specifica il nodo iniziale.
next	Attributo	Riferimento al nodo successivo.
prev	Attributo	Riferimento al nodo precedente.
up	Attributo	Riferimento al nodo superiore.
menu		Inserisce un menù Info manuale.
detailmenu	#PCDATA	Dettaglio nel menù Info.
intro		Delimita i capitoli che compongono l'introduzione.
h1		Titolo di un capitolo introduttivo.
h2		Titolo di una sezione introduttiva.
h3		Titolo di una sottosezione introduttiva.
h4		Titolo di una sotto-sottosezione introduttiva.
body		Delimita il corpo del documento.
tomeheading		Titolo di un tomo.
partheading		Titolo di una parte.
h1		Titolo di un capitolo.
h2		Titolo di una sezione.
h3		Titolo di una sottosezione.
h4		Titolo di una sotto-sottosezione.
appendix		Delimita i capitoli che compongono l'appendice.
h1		Titolo di un'appendice.
h2		Titolo di una sezione di appendice.
h3		Titolo di una sottosezione di appendice.
h4		Titolo di una sotto-sottosezione di appendice.
indexheading		Titolo di un indice analitico.

Elemento o attributo	Contenuto	Descrizione
printindex	Vuoto	Inserisce un indice analitico particolare.
name	Attributo	Sigla dell'indice analitico da inserire.
titolo_generico		I titoli hanno degli attributi in comune.
id	Attributo	Ancora per i riferimenti ipertestuali.
node	Attributo	Definizione manuale del nodo.
menu	Attributo	Titolo che appare nel menù.
next	Attributo	Definizione manuale del prossimo nodo.
prev	Attributo	Definizione manuale del nodo precedente.
up	Attributo	Definizione manuale del nodo superiore.
titolo_h		Dal capitolo in giù c'è un attributo aggiuntivo.
type	Attributo	Numerato, non numerato o intestazione semplice: '**numbered**', '**unnumbered**', '**heading**'.

Intestazione

L'elemento '**head**' è il più complicato. È necessario per definire molte informazioni che riguardano il documento. Segue un esempio abbastanza completo, che si riferisce alla documentazione ipotetica dello stesso Sgmltexi.

```
<head>
    <admin>
        <setfilename content="sgmltexi.info">
        <settitle content="Sgmltexi">
        <setchapternewpage content="odd">
        <defindex name="sg">
        <syncodeindex from="sg" to="cp">
        <infodir cat="Texinfo documentation system">
    </admin>
    <titlepage>
        <title>Sgmltexi</title>
        <subtitle>An alternative way to write Texinfo
        documentation</subtitle>
        <subtitle>This edition is for Sgmltexi
        &EDITION; (alpha) for Texinfo 4.0</subtitle>
        <abstract>
            <p>Sgmltexi is an SGML system (DTD and tools) to
            make Texinfo documentation using SGML...</p>
            ...
        </abstract>
        <author>Daniele Giacomini &lt;daniele@swlibero.org&gt;</author>
        <legal>
            <copyright>Copyright &copy; 2000 ...</copyright>
            <publishnote>
                <p>Published by...</p>
            </publishnote>
            <license>
                <p>Permission is granted to make and distribute
                verbatim copies of this manual...</p>
                ...
            </license>
            <coverart>
                <p>Cover art by ...</p>
            </coverart>
        </legal>
    </titlepage>
    <shortcontents>
    <contents>
</head>
```

Guardando l'esempio, si possono riconoscere alcuni elementi importanti: '**admin**', usato per alcune informazioni amministrative, e '**titlepage**'.

Informazioni amministrative

L'elemento '**admin**' viene usato per indicare al suo interno alcune informazioni che vanno prevalentemente nell'intestazione del documento Texinfo finale, oppure subito dopo. I componenti di questo ambiente non hanno un ordine preciso, nel sorgente SGML, in quanto poi vengono riordinati prima della composizione in Texinfo.

Nel seguito vengono elencati e descritti gli elementi che possono apparire all'interno di '**admin**'.

- '**setfilename**'

  Si tratta di un elemento vuoto, utilizzato per definire il nome del file Info finale, attraverso il comando '**@setfilename**' di Texinfo. Si usa con l'attributo '**content**' a cui si assegna il nome di questo file.

  ```
  <setfilename content="sgmltexi.info">
  ```

  L'esempio mostra il caso in cui si definisce il nome '`sgmltexi.info`'. Si può vedere che non serve il marcatore di chiusura.

- '**settitle**'

  Si tratta di un elemento vuoto, utilizzato per definire il titolo per la composizione in formato Info, attraverso il comando '**@settitle**' di Texinfo. Si usa con l'attributo '**content**' a cui si assegna questo titolo.

  ```
  <settitle content="Sgmltexi">
  ```

  L'esempio mostra il caso in cui si definisce il nome '**Sgmltexi**'. Si può vedere che non serve il marcatore di chiusura.

- '**setchapternewpage**'

  Si tratta di un elemento vuoto, non essenziale, utilizzato per definire il comando corrispondente di Texinfo: '**@setchapternewpage**'. Si assegna una parola chiave all'attributo '**content**', tra '**on**', '**off**' e '**odd**'.

  ```
  <setchapternewpage content="on">
  ```

  L'esempio mostra la richiesta esplicita di iniziare ogni capitolo in una pagina nuova.

  Il programma frontale di Sgmltexi, '**sgmltexi**', accetta un'opzione con lo stesso nome ('**--setchapternewpage=**{**on**|**off**|**odd**}' che prevale su quanto stabilito nel sorgente SGML in questo modo.

- '**footnotestyle**'

  Si tratta di un elemento vuoto, non essenziale, utilizzato per definire il comando corrispondente di Texinfo: '**@footnotestyle**'. Si assegna una parola chiave all'attributo '**content**', che può essere '**end**' o '**separate**'.

  ```
  <footnotestyle content="end">
  ```

  L'esempio mostra la richiesta esplicita di inserire i piè pagina alla fine della pagina a cui si riferiscono.

  Il programma frontale di Sgmltexi accetta un'opzione con lo stesso nome ('**--footnotestyle=**{**end**|**separate**}') che prevale su quanto stabilito nel sorgente SGML in questo modo.

- '**headings**'

  Si tratta di un elemento vuoto, non essenziale, utilizzato per definire il comando corrispondente di Texinfo: '**@headings**'. Si assegna una parola chiave all'attributo '**content**', che può essere: '**on**', '**off**', '**single**', '**double**', '**singleafter**', '**doubleafter**'.

  ```
  <headings content="on">
  ```

  L'esempio mostra la richiesta esplicita di mostrare le intestazioni.

  Il programma frontale di Sgmltexi accetta un'opzione con lo stesso nome, a cui si assegnano le stesse parole chiave ('**--headings=***impostazione*'), che prevale su quanto stabilito nel sorgente SGML in questo modo.

- '**defindex**', '**defcodeindex**'

  Si tratta di elementi vuoti, non essenziali, utilizzati per definire i comandi corrispondenti di Texinfo: '**@defindex**' e '**@defcodeindex**'. Si assegna un nome composto da due lettere all'attributo '**name**', per definire un indice analitico aggiuntivo; in particolare, utilizzando l'elemento '**defcodeindex**' si ottiene la creazione di un indice analitico composto da voci riprodotte in dattilografico.

  ```
  <defindex name="sg">
  ```

L'esempio mostra la definizione dell'indice analitico normale, identificato dalla sigla '**sg**'.

Naturalmente, si possono inserire più elementi '**defindex**' e '**defcodeindex**', quanti sono gli indici specifici che si vogliono dichiarare.

- '**synindex**', '**syncodeindex**'

  Questi due elementi vuoti, vengono usati per copiare le voci di un indice analitico all'interno di un altro, come fanno i comandi corrispondenti di Texinfo: '**@synindex**' e '**@syncodeindex**'. Questi due elementi richiedono l'indicazione di due attributi, '**from**' e '**to**', a cui si assegna rispettivamente la sigla dell'indice analitico di partenza e quella dell'indice di destinazione. Si osservi l'esempio:

```
<syncodeindex from="fn" to="cp">
```

  In questo caso, si trasferiscono tutte le voci dell'indice '**fn**' (quello delle funzioni) nell'indice '**cp**' (l'indice analitico standard). In particolare, dal momento che si tratta di '**syncodeindex**', le voci che vengono trasferite sono poi rese in modo dattilografico (con il comando '**@code**').

- '**infodir**'

  Questo elemento viene usato per definire una voce da inserire nell'elenco principale Info, quando il file relativo viene installato con il comando '**install-info**'. L'elemento contiene l'attributo '**cat**' a cui si assegna la categoria, come si fa con il comando '**@dircategory**' di Texinfo.

```
<infodir cat="Texinfo documentation system">
```

  L'elemento '**infodir**' può essere vuoto, come appena mostrato nell'esempio, ottenendo così l'inserimento di una sola riga nel corpo del comando '**@direntry**' di Texinfo, utilizzando le informazioni già conosciute: il nome del file Info e il titolo del documento. Se si vuole fare a mano, è possibile inserire queste informazioni all'interno dell'elemento, come nell'esempio seguente:

```
<infodir cat="Sistema di documentazione Sgmltexi">
* Sgmltexi: (sgmltexi).              Il mio bel manuale di Sgmltexi
* Introduzione: (sgmltexi)Intro 1.   Introduzione al sistema Sgmltexi
</infodir>
```

Pagine iniziali

L'elemento '**titlepage**' viene utilizzato per circoscrivere le informazioni che appaiono nelle primissime pagine del documento. L'ordine degli elementi contenuti è importante e gli errori vengono segnalati dal sistema di analisi SGML.

- '**title**'

  L'elemento '**title**' serve a contenere il titolo del documento nella sua forma stampata. Si traduce in Texinfo nel comando '**@title**'. Il suo utilizzo è molto semplice, come si vede dall'esempio seguente:

```
<title>Sgmltexi</title>
```

- '**subtitle**'

  Questo elemento permette l'indicazione di un sottotitolo. Non è obbligatorio e può essere usato più volte per indicare più sottotitoli successivi.

```
<subtitle>An alternate way to write Texinfo documentation</subtitle>
```

- '**abstract**'

  L'elemento '**abstract**' è facoltativo e si può usare una volta sola. Serve a racchiudere dei blocchi di testo, per esempio elementi '**p**', che descrivono in breve il contenuto del documento. Il contenuto di questo elemento viene utilizzato nella composizione Info, inserendolo nella parte iniziale del nodo '**top**'.

```
<abstract>
    <p>Sgmltexi is an SGML system (DTD and tools) to
    make Texinfo documentation using SGML...</p>
    ...
    <p>...</p>
</abstract>
```


- '**author**'

  Questo elemento, che deve essere indicato almeno una volta e può ripetersi a piacere, serve a contenere il nominativo di uno degli autori del documento. In Texinfo si traduce nel comando '**@author**'.

```
<author>Tizio Tizi &lt;tizio@dinkel.brot.dg&gt;</author>
<author>Caio Cai &lt;caio@dinkel.brot.dg&gt;</author>
```

  L'esempio mostra anche l'inclusione dell'indirizzo di posta elettronica, che comunque non sarebbe necessario.

- '**frontcovertext**'

  Questo elemento facoltativo, permette di inserire dei blocchi di testo all'interno della copertina.

- '**tpextra**'

  Questo elemento facoltativo, può essere usato in diverse situazioni all'interno delle pagine iniziali. Il suo scopo è quello di delimitare dei blocchi di testo che non hanno trovato una classificazione specifica.

  Per la precisione, questo elemento può apparire subito prima e subito dopo dell'elemento '**legal**', inoltre, se viene usato l'elemento '**dedications**', può essere aggiunto subito dopo di questo.

- '**legal**'

  L'elemento '**legal**' si articola a sua volta in altri elementi più dettagliati, allo scopo di descrivere tutto ciò che rappresenta gli aspetti legali del documento: il copyright, la nota sui diritti (concessi o esclusi), oltre ad altre informazioni amministrative legate all'edizione.

  – '**copyright**'

    Questo elemento serve a contenere l'indicazione relativa ai diritti di autore. Se nel tempo si sono succeduti diversi proprietari, l'elemento '**copyright**' può essere indicato più volte, in base alla necessità (in base a quanto concordato). Si osservi l'esempio seguente:

```
<copyright>Copyright &copy; 1987-1999 Tizio Tizi</copyright>
<copyright>Copyright &copy; 2000 Caio Cai</copyright>
```

  – '**publishnote**'

    L'elemento '**publishnote**', facoltativo, permette l'inclusione di blocchi di testo il cui scopo è quello di inserire informazioni relative alla pubblicazione. Si può usare in modo simile a quanto si vede nell'esempio seguente:

```
<publishnote>
    <p>Published by...</p>
    <p>...</p>
</publishnote>
```

  – '**license**'

    L'elemento '**license**' è fatto per contenere blocchi di testo che descrivono le condizioni con le quali è rilasciato il documento, che solitamente si rifanno a una licenza allegata da qualche parte (eventualmente in un'appendice).

```
<license>
    <p>Permission is granted to copy, distribute and/or
    modify this document under the terms of the GNU Free
    Documentation License, Version 1.1 or any later version
    published by the Free Software Foundation; with no
    Invariant Sections, with no Front-Cover Texts, and with
    no Back-Cover Texts. A copy of the license is included
    in the section entitled "GNU Free Documentation
    License".</p>
</license>
```

  – '**coverart**'

    L'elemento '**coverart**', facoltativo, consente di scrivere una nota su chi sia l'ideatore della copertina. In generale, se si usa Sgmltexi non ha senso preoccuparsi di una cosa del genere, dal momento che tutto viene guidato dallo schema SGML del DTD. Tuttavia, esiste la possibilità di fare questa annotazione ugualmente.

```
<coverart>
    <p>Cover art by ...</p>
</coverart>
```

  L'elemento '**legal**' può essere usato anche in modo più semplice, se la struttura prevista non soddisfa le esigenze reali. In

pratica, al posto degli elementi appena descritti, può contenere dei semplici blocchi di testo, come nell'esempio seguente:

```
<legal>
    <p>Copyright &copy; 2000 ...</p>

    <p>Published by...</p>

    <p>Permission is granted to make and distribute
    verbatim copies of this manual...</p>

    <p>Cover art by ...</p>
</legal>
```

- '**dedications**'

  Dopo l'elemento '**legal**', l'elemento '**dedications**' consente di elencare le dediche del documento. Queste appaiono esclusivamente nella composizione stampata, in una pagina apposita. L'elemento '**dedications**' è predisposto per l'inserimento di blocchi di testo di qualunque genere.

```
<dedications>
    <flushright>Ad Anna,<br>la mia amata.</flushright>
</dedications>
```

Indice generale

Dopo l'elemento '**titlepage**' è possibile collocare uno o più indici generali, più o meno dettagliati.

- '**contents**'

  L'elemento '**content**', vuoto, richiede l'inserimento di un indice generale dettagliato. Si traduce in pratica nel comando '**@content**' di Texinfo.

- '**shortcontents**', '**summarycontents**'

  Questi due elementi, vuoti, servono a includere rispettivamente i comandi '**@shortcontent**' e '**@summarycontent**' di Texinfo. Lo scopo è quello di ottenere un tipo di indice generale ridotto. Se si usa questo tipo di indice, si include solo uno dei due elementi in questione.

Nodi e menù Info iniziale

In mancanza di indicazioni, Sgmltexi gestisce da solo i collegamenti riferiti al nodo '**Top**', oltre a un menù unico per Info, collocato nello stesso nodo iniziale.

Volendo è possibile dichiarare espressamente il nodo '**Top**', attraverso l'elemento '**topnode**', che si usa vuoto con tre eventuali attributi: '**next**', '**prev**' e '**up**'. L'elemento '**topnode**' si colloca, eventualmente, subito dopo gli indici generali.

```
<topnode next="intro" prev="Top" up="(dir)">
```

Dopo l'elemento '**topnode**', è possibile specificare il menù iniziale in modo dettagliato, attraverso l'elemento '**menu**'. L'esempio seguente mostra un caso abbastanza articolato, benché abbreviato, in cui si vede anche l'inclusione dell'elemento '**detailmenu**':

```
<menu>
* Copying::                    Your rights.
* Overview::                   Texinfo in brief.
...
* Structuring::                How to create chapters, sections, subsections,
                                 appendices, and other parts.
* Nodes::                      How to write nodes.
...

<detailmenu>

 --- The Detailed Node Listing ---

Overview of Texinfo

* Reporting Bugs::             Submitting effective bug reports.
* Using Texinfo::              Create printed or online output.
* Info Files::                 What is an Info file?
...
</detailmenu>
</menu>
```

Naturalmente, non si tratta di elementi indispensabili, ma solo utili se si desidera avere il controllo della gestione dei nodi del documento che si ottiene.

Introduzione

Dopo l'elemento '**head**' ci può essere l'elemento '**intro**', il cui scopo è quello di definire uno spazio in cui i capitoli assumono il ruolo di sezioni introduttive, non numerate. Nell'ambito di questo spazio, i «capitoli» sono delimitati nello stesso modo utilizzato nel corpo del documento (l'elemento '**body**') e nelle appendici (l'elemento '**appendix**').

```
<intro>
<h1>Introduction to Sgmltexi</h1>

<p>Sgmltexi is a DTD with tools to get Texinfo...</p>

<p>Sgmltexi manage Texinfo nodes automatically,...</p>

</intro>
```

Corpo

Il corpo del documento è contenuto nell'elemento '**body**', che si colloca dopo l'elemento '**head**' e dopo l'elemento '**intro**' eventuale.

Il corpo può essere suddiviso in capitoli, oppure in parti, o anche in tomi, a seconda della dimensione del progetto di documentazione che si intende avviare. Lo spazio del tomo, della parte, del capitolo, o di una classificazione inferiore, non è delimitato esplicitamente, in quanto appare soltanto la dichiarazione del titolo, all'interno di un elemento che cambia a seconda del livello gerarchico. In pratica, il titolo di un tomo è racchiuso nell'elemento '**tomeheading**', mentre quello di una parte è inserito nell'elemento '**partheading**'.

I capitoli e le classificazioni inferiori hanno titoli delimitati da elementi analoghi a quelli dell'HTML: '**h1**', '**h2**', '**h3**' e '**h4**'. Questa classificazione, a partire da '**h1**' in giù, riguarda nello stesso modo l'introduzione e l'appendice.

```
<body>
<partheader>Networking</partheader>

<h1>IP protocol history</h1>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<h2>ISO-OSI model</h2>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<h1>IPv4 and IPv6</h1>

<p>Bla bla bla...</p>

<p>...</p>

</body>
```

Ogni elemento che racchiude un titolo consente l'inserimento dell'attributo '**id**', il cui scopo è quello di definire una stringa di identificazione, da usare come obiettivo per i riferimenti incrociati.

```
<h1 id="ip history">IP protocol history</h1>
```

È importante rammentare che, a causa di una limitazione progettuale di Texinfo, queste etichette per i riferimenti ipertestuali non possono contenere la virgola.

Ogni elemento che racchiude un titolo consente l'inserimento degli attributi '**node**' e '**menu**', con i quali è possibile stabilire il nome del nodo relativo e la descrizione che deve apparire nel menù (purché questo sia generato automaticamente). In mancanza di queste indicazioni, vengono generati dei nomi in modo automatico, mentre si usa il titolo come descrizione del nodo.

```
<h1 node="IPv4" menu="La storia del protocollo IP">Storia di IPv4</h1>
```

Ogni elemento che racchiude un titolo consente l'inserimento dell'attributo '**numbered**', a cui si possono assegnare esclusivamente le parole chiave '**on**' oppure '**off**'. In condizioni normali, l'attributo contiene la parola chiave '**on**', che implica la numerazione dei

titoli, salvo il caso dell'introduzione. Assegnando esplicitamente la parola chiave '**off**' si ottiene un titolo non numerato in un contesto che non lo prevederebbe.

```
<h1 numbered="off">Riconoscimenti</h1>
```

Ogni elemento che racchiude un titolo consente l'inserimento degli attributi '**next**', '**prev**' e '**up**'. Con questi si può alterare la catena di scorrimento dei nodi, specificandoli manualmente. In generale dovrebbe essere preferibile lasciare fare a Sgmltexi.

Appendice

Dopo il corpo del documento, delimitato dall'elemento '**body**', può apparire l'appendice, contenuta nell'elemento '**appendix**'. Al suo interno si possono inserire dei «capitoli», introdotti da un titolo contenuto in un elemento '**h1**', che vengono trattati correttamente come appendici. Dopo i titoli delimitati da '**h1**', sono ammissibili naturalmente anche segmenti di livello inferiore.

```
<appendix>
<h1>GNU Free Documentation License</h1>

<p indent="off"><strong>GNU Free Documentation License</strong></p>

<p indent="off">Version 1.1, March 2000</p>

<format>
Copyright &copy; 2000  Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
</format>
...
...
</appendix>
```

Indici analitici

Dopo il corpo e dopo il blocco delle appendici, è possibile inserire uno o più indici analitici. Questi si dichiarano con un titolo, attraverso l'elemento '**indexheading**' e con il riferimento al tipo di indice che si vuole esattamente, con l'elemento vuoto '**printindex**'. Si osservi l'esempio seguente in cui si inseriscono due indici: quello delle funzioni (la sigla '**fn**') e quello standard (la sigla '**cp**').

```
<indexheading>Index of functions</indexheading>
<printindex name="fn">
<indexheading>Concept index</indexheading>
<printindex name="cp">
```

Come si vede dall'esempio, l'elemento '**printindex**' ha l'attributo '**name**', a cui si assegna la sigla corrispondente all'indice che si vuole inserire.

Scomposizione del documento, nodi e menù Info

Per scrivere della documentazione di qualità, secondo i canoni di Texinfo, è necessario gestire direttamente i nodi e i menù. Con Sgmltexi si possono dimenticare i nodi e i menù, ma il risultato in formato Info potrebbe soffrirne. Tuttavia, come in parte è già stato mostrato, è possibile scegliere diversi livelli di automatismo in questa gestione.

Gli elementi usati per delimitare le intestazioni, da '**h1**' a '**h4**', possono incorporare gli attributi '**node**' e '**menu**'. Ciò prevale sulla determinazione automatica relativa. Si osservi l'esempio:

```
<h1 id="ip history" node="history" menu="History of IP protocol">
IP protocol history</h1>
```

In questo caso, si ottiene l'inserimento della riga seguente nel menù relativo:

```
* history::          History of IP protocol
```

I due attributi, '**node**' e '**menu**', possono essere usati in modo indipendente: l'attributo che non viene usato, viene sostituito in modo automatico.

> Avendo accesso ai nodi, è possibile farvi riferimento per dei riferimenti incrociati, senza bisogno di usare l'attributo '**id**'.

Come già descritto in precedenza, Sgmltexi crea automaticamente il nodo '**Top**' iniziale. Il menù relativo può essere definito esplicitamente e in tal caso tutti i nodi e tutte le descrizioni relative devono essere inseriti manualmente.

Inserendo l'elemento '**menu**' alla fine del testo di un capitolo, o di una sezione inferiore, si ottiene l'aggiunta di un menù Info in corrispondenza di quel punto. Si osservi l'esempio:

```
<h1>IP protocol history</h1>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<menu>

<h2>ISO-OSI model</h2>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<h2>More information</h2>

<p>Bla bla bla...</p>

<p>...</p>
```

In questo caso, si ottiene l'inserzione di un menù, gestito automaticamente, prima delle sezioni di livello '**h2**'. Volendo, si può indicare il menù in modo preciso, come si vede di seguito:

```
<menu>
* IP layer::        IP ISO-OSI layer model
* more on IP::      More details on IP
</menu>
```

Quando un menù viene descritto in questo modo, i nomi dei nodi devono essere identici a quelli dichiarati negli elementi delle intestazioni. In pratica, scrivendo un menù in modo manuale, anche i nodi devono essere dichiarati esattamente, come si vede qui:

```
<h1>IP protocol history</h1>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<menu>
* IP layer::        IP ISO-OSI layer model
* more on IP::      More details on IP
</menu>

<h2 node="IP layer">ISO-OSI model</h2>

<p>Bla bla bla...</p>

<p>Bla bla bla...</p>

<h2 node="more on IP">More information</h2>

<p>Bla bla bla...</p>

<p>...</p>
```

È evidente, in questa situazione, che l'attributo '**menu**', il cui scopo sarebbe quello di controllare la descrizione del nodo nel menù, non può essere preso in considerazione in questo caso.

Numerazione o meno dei titoli

Texinfo consente di inserire dei titoli riferiti a capitoli o sezioni inferiori, con o senza numerazione. Inoltre, consente anche di dichiarare dei titoli che non devono apparire nell'indice generale. Per controllare questa possibilità con Sgmltexi, si può utilizzare l'attributo '**type**' che riguarda tutti gli elementi '**h*n***':

```
<hn type="{numbered|unnumbered|heading}">titolo</hn>
```

In mancanza dell'indicazione dell'attributo, è come se gli fosse stata assegnata la parola chiave '**numbered**', con la quale i titoli del corpo e delle appendici sono numerati (con numeri o lettere rispettivamente). Utilizzando la parola chiave '**numbered**' si ottiene l'inserimento di un titolo non numerato (nel caso dell'introduzione è sempre senza numerazione); con la parola chiave '**heading**' si ottiene un titolo non numerato e anche non segnalato nell'indice generale (in questo senso può essere utile anche nell'introduzione).

Codifica

Sgmltexi ha una gestione incompleta per le codifiche ISO 8859-***n***. È incompleta perché Texinfo non è in grado di riprodurre tutti i caratteri. Ci sono due modi per definire l'uso di una codifica particolare con Sgmltexi: l'opzione '**`--input-encoding`**' e l'attributo '**`charset`**' all'interno dell'elemento '**`sgmltexi`**'.

La scelta genera risultati differenti. L'opzione '**`--input-encoding`**' genera una trasformazione dei caratteri in entità SGML, che successivamente sono tradotte in codice Texinfo. In questo modo, il codice Texinfo che si ottiene è sicuramente in ASCII puro (ISO 646), dove le entità che non hanno alcuna corrispondenza in Texinfo. vengono mostrate come '**`[ETH ]`**', tanto per fare un esempio. L'uso dell'attributo '**`charset`**' si traduce semplicemente nel comando '**`@documentencoding`**'; in certe situazioni, il risultato della composizione può essere buono o meno. A seconda del risultato migliore che si riesce a ottenere, si può scegliere un modo invece dell'altro.

Una buona strategia può essere l'uso dell'attributo '**`charset`**' in ogni caso, aggiungendo l'opzione '**`--input-encoding`**' quando Texinfo non genera una composizione piacevole (di solito quando si genera un formato per la stampa).

Entità standard e non standard

Il DTD di Sgmltexi include tutte le entità standard ISO 8879. Tuttavia, non tutte le entità sono gestibili da Texinfo; pertanto, quando si usa un'entità non gestibile, viene mostrata nella composizione finale come racchiusa tra parentesi quadre, per esempio come '**`[ETH ]`**'.

Sgmltexi mette a disposizione qualche entità non standard, necessaria per mantenere la compatibilità con Texinfo. Queste entità speciali sono elencate nella tabella u90.41.

Tabella u90.41. Entità non standard.

Macro SGML	Comando Texinfo	Descrizione
`&dots;`	`@dots{}`	Tre puntini.
`&enddots;`	`@enddots{}`	Quattro puntini.
`&TeX;`	`@TeX{}`	Il nome «TeX»
`&result;`	`@result{}`	
`&expansion;`	`@expansion{}`	
`&print;`	`@print{}`	
`&error;`	`@error{}`	
`&point;`	`@point{}`	
`&today;`	`@today{}`	
`&esexcl;`	`@!`	Punto esclamativo alla fine di una frase.
`&esperiod;`	`@.`	Punto fermo alla fine di una frase.
`&nes;`	`@:`	Frase che non si conclude.
`&esquest;`	`@?`	Punto interrogativo alla fine di una frase.

Sgmltexi: contenuti

Paragrafi 641
Indici e riferimenti incrociati 641
Delimitazione di parole e di frasi 643
Delimitazione di blocchi di testo 644
Elenchi e tabelle 645
Inserzioni 647
Definizioni 647
Codice condizionato e codice letterale in base alla composizione 649
Problemi 651

Dopo la struttura generale, il sorgente Sgmltexi si articola generalmente in elementi che possono essere classificati sommariamente in blocchi e in testo interno a un blocco. Nei DTD comuni si utilizzano frequentemente le entità parametriche '**`%block;`**' e '**`%inline;`**', per definire questi due grandi raggruppamenti. Nel DTD di Sgmltexi si usa la stessa convenzione e in questo senso vanno interpretate tali sigle nelle tabelle riassuntive.

A titolo di esempio, un blocco è qualcosa di simile a un paragrafo, un elenco, una tabella; un elemento interno alla riga è fatto per contenere del testo, eventualmente assieme a delle enfatizzazioni di qualche genere. Di solito, anche se questo fatto non può valere in generale, un elemento interno alla riga è fatto per contenere testo o altri elementi dello stesso genere; al contrario, un elemento che costituisce un blocco, può contenere altri blocchi, oppure del testo interno alla riga.

Il DTD di Sgmltexi non prevede elementi che possano contenere testo interno alla riga o blocchi a scelta, come accade invece nell'HTML.

Paragrafi

I blocchi di testo più comuni sono dei paragrafi, delimitati dall'elemento '**`p`**', il quale può apparire con un rientro iniziale o meno, a seconda dell'uso dell'attributo '**`indent`**'. I paragrafi, compresi quelli centrati che si ottengono con l'elemento '**`center`**', contengono testo o altri elementi interni alla riga.

Tabella u91.1. Paragrafi con Sgmltexi.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
p	%inline;	Blocco di testo elementare, riconducibile al paragrafo.
indent	Attributo	Rientro prima riga: '**`on`**', '**`off`**'.
center	%inline;	Blocco di testo centrato: '**`@center`**'.

Indici e riferimenti incrociati

Sgmltexi mette a disposizione diversi elementi il cui scopo è quello di permettere delle inserzioni per generare degli indici o dei riferimenti incrociati, riproducendo i comandi equivalenti di Texinfo.

Le voci degli indici analitici vengono inserite attraverso un gruppo di elementi vuoti: '**`cindex`**', '**`findex`**', '**`vindex`**', '**`kindex`**', '**`pindex`**', '**`tindex`**' e '**`userindex`**'. Tutti questi elementi hanno lo stesso attributo '**`entry`**', che serve a specificare la voce da inserire nell'indice relativo. In particolare, l'elemento '**`userindex`**' ha in più l'attributo '**`name`**' per specificare l'indice al quale si vuole fare riferimento.

Questi elementi possono essere usati solo dopo la dichiarazione di una sezione (un titolo di qualunque tipo, dal tomo in giù), ma prima del testo normale che ne seguirebbe. Per esempio così:

```
<h1>IP protocol history</h1>
<cindex entry="IP protocol">
<cindex entry="history">

<p>Bla bla bla...</p>
```

La tabella u91.3 riassume brevemente l'uso di questi elementi.

Tabella u91.3. Voci degli indici analitici.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
cindex	Vuoto	Voce dell'indice analitico normale.
entry	Attributo	Voce da inserire nell'indice.
findex	Vuoto	Voce dell'indice analitico delle funzioni.
entry	Attributo	Voce da inserire nell'indice.
vindex	Vuoto	Voce dell'indice analitico delle variabili.
entry	Attributo	Voce da inserire nell'indice.
kindex	Vuoto	Voce dell'indice analitico dei tasti premuti.
entry	Attributo	Voce da inserire nell'indice.
pindex	Vuoto	Voce dell'indice analitico dei programmi.
entry	Attributo	Voce da inserire nell'indice.
tindex	Vuoto	Voce dell'indice analitico dei tipi di dati.
entry	Attributo	Voce da inserire nell'indice.
userindex	Vuoto	Voce di un indice analitico definito dall'utilizzatore.
entry	Attributo	Voce da inserire nell'indice.
name	Attributo	Sigla identificativa dell'indice definito dall'utente.
printindex	Vuoto	Inserisce l'elenco delle voci dell'indice specificato.
name	Attributo	Sigla identificativa dell'indice.

Ogni indice analitico si distingue in base a una sigla di due lettere. Gli indici analitici già previsti da Texinfo hanno una sigla fissa, mentre tutte le altre combinazioni possono essere usate per gli indici stabiliti dall'utilizzatore. La tabella u91.4 riassume le sigle degli indici standard, la cui conoscenza è necessaria per poter usare correttamente l'elemento '**`printindex`**' allo scopo di riprodurre l'elenco dell'indice relativo.

Tabella u91.4. Sigle identificative degli indici analitici standard.

Sigla	Descrizione
cp	Indice analitico normale.
ky	Indice analitico dell'uso della tastiera.
pg	Indice analitico dei programmi.
fn	Indice analitico delle funzioni.
vr	Indice analitico delle variabili.
tp	Indice analitico dei tipi di dati.

Gli elementi utilizzati per realizzare dei riferimenti incrociati sono vuoti e sono sempre interni alla riga di testo. Tutte le informazioni necessarie sono passate attraverso attributi. Dal momento che questi elementi rispecchiano fedelmente i comandi equivalenti di Texinfo, viene mostrata solo la tabella u91.5, senza entrare nel dettaglio del significato di ognuno di loro.

Tabella u91.5. Riferimenti incrociati.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
anchor	Vuoto	Comando '**`@anchor`**' di Texinfo.
id	Attributo	Stringa di identificazione dell'ancora.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
xref	Vuoto	Comando '**`@xref`**' di Texinfo.
id	Attributo	Nodo o ancora a cui si fa riferimento.
name	Attributo	Nome del riferimento.
title	Attributo	Titolo o argomento a cui si fa riferimento.
info	Attributo	Nome del file Info.
ptitle	Attributo	Titolo dell'edizione stampata.
ref	Vuoto	Comando '**`@ref`**' di Texinfo.
id	Attributo	Nodo o ancora a cui si fa riferimento.
name	Attributo	Nome del riferimento.
title	Attributo	Titolo o argomento a cui si fa riferimento.
info	Attributo	Nome del file Info.
ptitle	Attributo	Titolo dell'edizione stampata.
pxref	Vuoto	Comando '**`@pxref`**' di Texinfo.
id	Attributo	Nodo o ancora a cui si fa riferimento.
name	Attributo	Nome del riferimento.
title	Attributo	Titolo o argomento a cui si fa riferimento.
info	Attributo	Nome del file Info.
ptitle	Attributo	Titolo dell'edizione stampata.
inforef	Vuoto	Comando '**`@inforef`**' di Texinfo.
id	Attributo	Nodo o ancora a cui si fa riferimento.
name	Attributo	Nome del riferimento.
info	Attributo	Nome del file Info.
uref	Vuoto	Comando '**`@uref`**' di Texinfo.
uri	Attributo	Indirizzo URI a cui si fa riferimento.
name	Attributo	Nome del riferimento.
replace	Attributo	Testo di rimpiazzo da mostrare.
email	Vuoto	Comando '**`@email`**' di Texinfo.
email	Attributo	Indirizzo di posta elettronica.
name	Attributo	Titolo o descrizione dell'indirizzo.

In particolare, è opportuno osservare che l'attributo '**`id`**' degli elementi '**`h`*n***', '**`partheading`**' e '**`tomeheading`**', è un'ancora a cui possono puntare tutti i vari tipi di riferimenti incrociati disponibili (tranne '**`uref`**' e '**`email`**' che puntano a degli URI).

L'esempio seguente mostra come usare l'elemento '**`pxref`**' in modo molto semplice:

```
<p>Sgmltexi crea automaticamente il nodo Top. Come già spiegato in
precedenza, (<pxref id="top node menu">), il menù può essere...</p>
```

Delimitazione di parole e di frasi

Un certo numero di elementi serve a delimitare parole o frasi, per qualche motivo. Il DTD di Sgmltexi è molto permissivo, in modo tale che ogni elemento di questi può contenere qualunque altro elemento interno alla riga di testo. Ciò è stato fatto per assicurare la massima compatibilità con Texinfo, ma in futuro potrebbero essere poste delle piccole limitazioni.

La tabella u91.7 elenca questi elementi, assieme a '**`kbdinputstyle`**', che si usa per specificare lo stile di rappresentazione del contenuto dell'elemento '**`kbd`**'.

Tabella u91.7. Delimitazione di parole e frasi.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
code	%inline;	Comando '**`@code`**' di Texinfo.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
kbd	%inline;	Comando ‘**@kbd**’ di Texinfo.
kbdinputstyle	Vuoto	Comando ‘**@kbdinputstyle**’ di Texinfo.
style	Attributo	Stile: ‘**code**’, ‘**example**’, ‘**distinct**’.
key	%inline;	Comando ‘**@key**’ di Texinfo.
samp	%inline;	Comando ‘**@samp**’ di Texinfo.
var	%inline;	Comando ‘**@var**’ di Texinfo.
env	%inline;	Comando ‘**@env**’ di Texinfo.
file	%inline;	Comando ‘**@file**’ di Texinfo.
command	%inline;	Comando ‘**@command**’ di Texinfo.
option	%inline;	Comando ‘**@option**’ di Texinfo.
dfn	%inline;	Comando ‘**@dfn**’ di Texinfo.
cite	%inline;	Comando ‘**@cite**’ di Texinfo.
acronym	%inline;	Comando ‘**@acronym**’ di Texinfo.
url	%inline;	Comando ‘**@url**’ di Texinfo.
emph	%inline;	Comando ‘**@emph**’ di Texinfo.
strong	%inline;	Comando ‘**@strong**’ di Texinfo.
sc	%inline;	Comando ‘**@sc**’ di Texinfo.
roman	%inline;	Comando ‘**@r**’ di Texinfo.
italic	%inline;	Comando ‘**@i**’ di Texinfo.
bold	%inline;	Comando ‘**@b**’ di Texinfo.
typewriter	%inline;	Comando ‘**@t**’ di Texinfo.

Viene mostrato un esempio molto semplice dell’uso dell’elemento ‘**strong**’:

```
<p><strong>Pinco Pallino</strong> è un uomo molto vecchio...</p>

<p><strong>Tizio Tizi</strong> ha studiato tecnologia delle
comunicazioni...</p>
```

Delimitazione di blocchi di testo

Alcuni elementi servono a delimitare blocchi di testo, o un tipo particolare di testo interno alle righe. Il DTD di Sgmltexi è molto permissivo per assicurare la massima compatibilità con Texinfo, ma in futuro potrebbero essere poste delle piccole limitazioni.

La tabella u91.9 elenca questi elementi, assieme a ‘**pre**’, che permette di inserire del testo preformattato, e a ‘**exdent**’, utilizzato all’interno di ‘**pre**’ per ottenere delle righe che sporgono verso l’esterno.

Tabella u91.9. Delimitazione di blocchi di testo.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
exdent	%inline;	Comando ‘**@exdent**’ di Texinfo.
pre	%inline;	Testo preformattato.
quotation	%block;	Comando ‘**@quotation**’ di Texinfo.
display	%block; o ‘**pre**’	Comando ‘**@display**’ di Texinfo.
smalldisplay	%block; o ‘**pre**’	Comando ‘**@smalldisplay**’ di Texinfo.
example	%block; o ‘**pre**’	Comando ‘**@example**’ di Texinfo.
smallexample	%block; o ‘**pre**’	Comando ‘**@smallexample**’ di Texinfo.
flushleft	%inline;	Comando ‘**@flushleft**’ di Texinfo.
flushright	%inline;	Comando ‘**@flushright**’ di Texinfo.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
lisp	%block; o ‘**pre**’	Comando ‘**@lisp**’ di Texinfo.
smalllisp	%block; o ‘**pre**’	Comando ‘**@smalllisp**’ di Texinfo.
cartouche	%block; o ‘**pre**’	Comando ‘**@cartouche**’ di Texinfo.
format	%block; o ‘**pre**’	Comando ‘**@format**’ di Texinfo.
smallformat	%block; o ‘**pre**’	Comando ‘**@smallformat**’ di Texinfo.
texinfo		Codice Texinfo incorporato.

In generale, l’uso di questi elementi è molto semplice, come si può vedere in questo caso:

```
<example>
<p>Bla bla bla...</p>
<p>Bla bla bla...</p>
</object>
```

L’esempio seguente, invece, mostra l’uso dell’elemento ‘**pre**’, allo scopo di incorporare del testo preformattato, pur continuando a espandere le macro SGML:

```
<example>
<pre>
#!/usr/bin/perl
while ($line = &gt;STDIN&gt;)
  {
    chomp $line;
    print ("$line\r\n");
  }
</pre>
</object>
```

In aggiunta, si può delimitare il contenuto dell’elemento ‘**pre**’ per poterlo scrivere in modo letterale:

```
<example>
<pre>
<![CDATA[
#!/usr/bin/perl
while ($line = <STDIN>)
  {
    chomp $line;
    print ("$line\r\n");
  }
]]>
</pre>
</example>
```

Elenchi e tabelle

Elenchi e tabelle, sono blocchi di testo. La gestione di Texinfo per ciò che riguarda queste strutture, è abbastanza speciale. Qui viene riassunto tutto nella tabella u91.13, che però richiede la conoscenza dei comandi di Texinfo corrispondenti.

Tabella u91.13. Elenchi e tabelle.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
itemize	‘**item**’, ‘**itemx**’, %block;	Comando ‘**@itemize**’ di Texinfo.
mark	Attributo	Segno usato al posto del pallino iniziale.
enumerate	‘**item**’, ‘**itemx**’, %block;	Comando ‘**@enumerate**’ di Texinfo.
start	Attributo	Valore iniziale dell’elenco numerato.
table	‘**item**’, ‘**itemx**’, %block;	Comando ‘**@table**’ di Texinfo.
emphasis	Attributo	Enfasi della colonna descrittiva: ‘**asis**’, ‘**code**’, ‘**samp**’, ‘**var**’, ‘**kbd**’, ‘**file**’.
vtable	‘**item**’, ‘**itemx**’, %block;	Comando ‘**@vtable**’ di Texinfo.
emphasis	Attributo	Enfasi della colonna delle variabili: ‘**asis**’, ‘**code**’, ‘**samp**’, ‘**var**’, ‘**kbd**’, ‘**file**’.
ftable	‘**item**’, ‘**itemx**’, %block;	Comando ‘**@ftable**’ di Texinfo.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
emphasis	Attributo	Enfasi della colonna delle funzioni: ‘**asis**’, ‘**code**’, ‘**samp**’, ‘**var**’, ‘**kbd**’, ‘**file**’.
item	%inline; o vuoto.	Comando ‘**@item**’ di Texinfo.
itemx	%inline; o vuoto.	Comando ‘**@itemx**’ di Texinfo.
multitable		Comando ‘**@multitable**’ di Texinfo.
columnfraction	‘**.*n***’.	Colonna larga 0,***n*** volte lo spazio totale.
columnexample	Testo puro.	Colonna larga tanto quanto l’esempio.
raw	%inline;, ‘**tab**’.	Riga di una tabella.
tab	Vuoto	Separatore tra una colonna e la successiva.

Vengono mostrati alcuni esempi, a cominciare da un elenco non numerato:

```
<itemize mark="#">
<item>
    <p>Primo elemento dell'elenco.</p>
<item>
    <p>Secondo elemento.</p>
</itemize>
```

In questo caso, si ottiene un elenco puntato di due sole voci, dove al posto del pallino usuale, appare il simbolo ‘#’. Sostituendo l’elemento ‘**itemize**’ con ‘**enumerate**’, si ottiene un elenco numerato:

```
<enumerate start="3">
<item>
    <p>Primo elemento dell'elenco.</p>
<item>
    <p>Secondo elemento.</p>
</enumerate>
```

In questo caso, si fa in modo che il primo dei due elementi abbia il numero tre. L’elenco descrittivo si ottiene attraverso l’elemento ‘**table**’, dove gli elementi ‘**item**’ contengono le voci relative. Si osservi l’esempio:

```
<table emphasis="code">
<item>ls</item>
<itemx>dir</itemx>
    <p>Elenco del contenuto della directory.</p>
<item>cd</item>
    <p>Cambia directory.</p>
</table>
```

Si intende così che l’elemento ‘**itemx**’ serve quando un elemento dell’elenco è composto da più di una voce.

Le tabelle, intese come quelle a cui si è abituati di solito, sono gestite attraverso l’elemento ‘**multitable**’. Questo, prima dell’indicazione delle righe che compongono la tabella, richiede di specificare quante sono le colonne e quanto larghe devono essere. Per questo, all’inizio occorre utilizzare una serie di elementi ‘**columnfraction**’, oppure ‘**columnexample**’, attraverso i quali si specificano proprio queste larghezze (in percentuale o attraverso un testo di esempio). L’esempio seguente mostra il caso di una tabella le cui colonne sono state definite in modo percentuale:

```
<multitable>
<columnfraction>.30</columnfraction>
<columnfraction>.70</columnfraction>
<raw><strong>Parametro LOC</strong>
<tab><strong>Posizione corrispondente</strong>
</raw>
<raw>h
<tab>posizione attuale
</raw>
<raw>t
<tab>superiore
</raw>
<raw>b
<tab>inferiore
</raw>
<raw>p
<tab>pagina
</raw>
</multitable>
```

In alternativa, dato che la larghezza delle colonne dipende proprio dai titoli, si potrebbe fare così:


```
<multitable>
<columnexample>Parametro LOC</columnexample>
<columnexample>Posizione corrispondente</columnexample>
<raw><strong>Parametro LOC</strong>
<tab><strong>Posizione corrispondente</strong>
</raw>
<raw>h
<tab>posizione attuale
</raw>
<raw>t
<tab>superiore
</raw>
<raw>b
<tab>inferiore
</raw>
<raw>p
<tab>pagina
</raw>
</multitable>
```

In entrambi i casi, lo scopo è quello di ottenere uno specchietto simile a quello che segue. Si osservi che non ci sono didascalie e nemmeno esiste la possibilità di collocare dinamicamente la tabella.

Parametro LOC	Posizione corrispondente
h	posizione attuale
t	superiore
b	inferiore
p	pagina

Inserzioni

Alcuni elementi sono difficilmente classificabili in gruppi particolari. Qui, vengono distinti in due raggruppamenti: quelli interni alle righe e quelli che rappresentano dei blocchi. A questi corrispondono le tabelle u91.20 e u91.21.

Tabella u91.20. Inserzioni interne alle righe.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
dmn	#PCDATA	Comando ‘**@dmn**’ di Texinfo.
math	#PCDATA	Comando ‘**@math**’ di Texinfo.
footnote	%inline;	Comando ‘**@footnote**’ di Texinfo.
image	Vuoto	Comando ‘**@image**’ di Texinfo.
name	Attributo	Nome del file da inserire, senza estensione.
width	Attributo	Ampiezza dell’immagine.
height	Attributo	Altezza dell’immagine.
whole	%inline;	Comando ‘**@w**’ (previene l’interruzione di riga).
br	Vuoto	Comando ‘**@***’ (interruzione di riga).
dh	Vuoto	Comando ‘**@-**’ (separazione facoltativa).
hyphenation	Vuoto;	Comando ‘**@hyphenation**’ di Texinfo.
words	Attributo	Elenco di parole separate in sillabe.

Tabella u91.21. Inserzione di blocchi.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
sp	Vuoto	Comando ‘**@sp**’ di Texinfo.
lines	Attributo	Quantità di righe da saltare (un numero intero).
page	Vuoto	Comando ‘**@page**’ di Texinfo.
group	%block;	Comando ‘**@group**’ di Texinfo.
need	Vuoto	Comando ‘**@need**’ di Texinfo.
mils	Attributo	Millesimi di pollice richiesti.

Definizioni

Texinfo prevede un grande numero di comandi per la descrizione di definizioni di vario genere. Queste «definizioni» vanno intese generalmente come dei modelli sintattici. È un po’ difficile comprende-

re bene quando usare questa o quella forma di definizione; per cui occorre studiare la documentazione di Texinfo.

Tutte le forme di definizione si dichiarano attraverso un elemento provvisto di diversi attributi. Questo elemento contiene generalmente la descrizione del modello, in una serie di blocchi di testo, ma in particolare potrebbe contenere la descrizione degli argomenti, all'interno dell'elemento '**args**', comune a tutte le definizioni che ne hanno.

Tabella u91.22. Definizioni.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
args	%inline;	Argomenti di una definizione.
deffn	'**args**', %block;	Comando '**@deffn**' di Texinfo.
cat	Attributo	Categoria della funzione.
name	Attributo	Nome della funzione.
deffnx	Vuoto	Comando '**@deffnx**'. Attributi come '**deffn**'.
defun	'**args**', %block;	Comando '**@defun**' di Texinfo.
name	Attributo	Nome della funzione.
defunx	Vuoto	Comando '**@defunx**'. Attributi come '**defun**'.
defmac	'**args**', %block;	Comando '**@defmac**' di Texinfo.
name	Attributo	Nome della macro.
defmacx	Vuoto	Comando '**@defmacx**'. Attributi come '**defmac**'.
defspec	'**args**', %block;	Comando '**@defspec**' di Texinfo.
name	Attributo	Nome di uno *special form*.
defspecx	Vuoto	Comando '**@defspecx**'. Attributi come '**defspec**'.
defvr	%block;	Comando '**@defvr**' di Texinfo.
cat	Attributo	Categoria della variabile.
name	Attributo	Nome della variabile.
defvrx	Vuoto	Comando '**@defvrx**'. Attributi come '**defvr**'.
defvar	%block;	Comando '**@defvar**' di Texinfo.
name	Attributo	Nome della variabile.
defvarx	Vuoto	Comando '**@defvarx**'. Attributi come '**defvar**'.
defopt	%block;	Comando '**@defopt**' di Texinfo.
name	Attributo	Nome dell'opzione.
defoptx	Vuoto	Comando '**@defoptx**'. Attributi come '**defopt**'.
deftypefn	'**args**', %block;	Comando '**@deftypefn**' di Texinfo.
cat	Attributo	Categoria.
type	Attributo	Tipo di dati.
name	Attributo	Nome.
deftypefnx	Vuoto	Comando '**@deftypefnx**'. Attributi come '**deftypefn**'.
deftypefun	'**args**', %block;	Comando '**@deftypefun**' di Texinfo.
type	Attributo	Tipo di dati.
name	Attributo	Nome.
deftypefunx	Vuoto	Comando '**@deftypefunx**'. Attributi come '**deftypefun**'.
deftypevr	%block;	Comando '**@deftypevr**' di Texinfo.
cat	Attributo	Categoria.
type	Attributo	Tipo di dati.
name	Attributo	Nome.
deftypevrx	Vuoto	Comando '**@deftypevrx**'. Attributi come '**deftypevr**'.
deftypevar	%block;	Comando '**@deftypevar**' di Texinfo.
type	Attributo	Tipo di dati.
name	Attributo	Nome.
deftypevarx	Vuoto	Comando '**@deftypevarx**'. Attributi come '**deftypevar**'.
args	%inline;	Argomenti di una definizione.
defcv	%block;	Comando '**@defcv**' di Texinfo.
cat	Attributo	Categoria.
class	Attributo	Classe.
name	Attributo	Nome.
defcvx	Vuoto	Comando '**@defcvx**'. Attributi come '**defcv**'.
defivar	%block;	Comando '**@defivar**' di Texinfo.
class	Attributo	Classe.
name	Attributo	Nome.
defivarx	Vuoto	Comando '**@defivarx**'. Attributi come '**defivar**'.
deftypeivar	%block;	Comando '**@deftypeivar**' di Texinfo.
class	Attributo	Classe.
type	Attributo	Tipo.
name	Attributo	Nome.
deftypeivarx	Vuoto	Comando '**@deftypeivarx**'. Attributi come '**deftypeivar**'.
defop	'**args**', %block;	Comando '**@defop**' di Texinfo.
cat	Attributo	Categoria.
class	Attributo	Classe.
name	Attributo	Nome.
defopx	Vuoto	Comando '**@defopx**'. Attributi come '**defop**'.
defmethod	'**args**', %block;	Comando '**@defmethod**' di Texinfo.
class	Attributo	Classe.
name	Attributo	Nome.
defmethodx	Vuoto	Comando '**@defmethodx**'. Attributi come '**defmethod**'.
deftypemethod	'**args**', %block;	Comando '**@deftypemethod**' di Texinfo.
class	Attributo	Classe.
type	Attributo	Tipo.
name	Attributo	Nome.
deftypemethodx	Vuoto	Comando '**@deftypemethodx**'. Attributi come '**deftypemethod**'.
deftp	'**args**', %block;	Comando '**@deftp**' di Texinfo.
cat	Attributo	Categoria.
name	Attributo	Nome.
deftpx	Vuoto	Comando '**@deftpx**'. Attributi come '**deftp**'.

Ecco un esempio molto semplice:

```
<deffn cat="Command" name="sgmltexi">
    <args>[<var>options</var>]... <var>sgml_source</var></args>

    <p>This is the front-end for the SGML to Texinfo system.</p>

</deffn>
```

La composizione in formato Info genera il risultato seguente:

```
 - Command: sgmltexi [OPTIONS]... SGML_SOURCE
     This is the front-end for the SGML to Texinfo system.
```

Codice condizionato e codice letterale in base alla composizione

Texinfo ha la possibilità di selezionare del codice in dipendenza del tipo di composizione finale. In SGML si possono fare cose simili attraverso le sezioni marcate, ma non si tratta della stessa cosa. Per

questa ragione, Sgmltexi include alcuni elementi speciali corrispondenti ai comandi che servono a Texinfo per selezionare il codice, consentendo anche di inserire pezzi di codice letterale.

Tabella u91.25. Codice condizionato e codice letterale in base alla composizione.

Elemento o attributo	Contenuto	Descrizione o corrispondenza con Texinfo
ifinfo	%inline;	'**@ifinfo**' ... '**@end ifinfo**'
ifinfoblock	%block;	'**@ifinfo**' ... '**@end ifinfo**'
iftex	%inline;	'**@iftex**' ... '**@end iftex**'
iftexblock	%block;	'**@iftex**' ... '**@end iftex**'
ifhtml	%inline;	'**@ifhtml**' ... '**@end ifhtml**'
ifhtmlblock	%block;	'**@ifhtml**' ... '**@end ifhtml**'
ifnotinfo	%inline;	'**@ifnotinfo**' ... '**@end ifnotinfo**'
ifnotinfoblock	%block;	'**@ifnotinfo**' ... '**@end ifnotinfo**'
ifnottex	%inline;	'**@ifnottex**' ... '**@end ifnottex**'
ifnottexblock	%block;	'**@ifnottex**' ... '**@end ifnottex**'
ifnothtml	%inline;	'**@ifnothtml**' ... '**@end ifnothtml**'
ifnothtmlblock	%block;	'**@ifnothtml**' ... '**@end ifnothtml**'
tex	#PCDATA	'**@tex**' ... '**@end tex**'
html	#PCDATA	'**@html**' ... '**@end html**'
texinfo	#PCDATA	Codice Texinfo.

È importante osservare che '**ifinfo**', '**iftex**', '**ifhtml**', '**ifnotinfo**', '**ifnottex**' e '**ifnothtml**', sono elementi interni alla riga di testo, che contengono lo stesso genere di cosa. Al contrario, '**ifinfoblock**', '**iftexblock**', '**ifhtmlblock**', '**ifnotinfoblock**', '**ifnottexblock**' e '**ifnothtmlblock**', sono blocchi che contengono altri blocchi. Questa distinzione è necessaria per evitare problemi nella definizione del documento SGML (nel DTD).

In particolare, gli elementi '**tex**', '**html**' e '**texinfo**', sono fatti per contenere testo letterale solitamente racchiuso tra '**<![CDATA[**' e '**]]>**'.

L'elemento '**texinfo**' non ha un comando equivalente in Texinfo, perché rappresenta del codice Texinfo. Si osservi l'esempio seguente:

```
<p>The letter <texinfo>@ubaraccent{o}</texinfo> is a special...</p>
```

Usando questo elemento, potrebbe essere necessario forzare l'interpretazione letterale anche da parte dell'SGML. In tal caso, il contenuto dell'elemento può essere racchiuso come si vede qui:

```
<p>The letter <texinfo><![CDATA[@ubaraccent{o}]]></texinfo> is a...
```

Il caso particolare dell'esempio non mostra una situazione in cui sia indispensabile l'interpretazione SGML letterale, tuttavia questo è il modo quando succede tale circostanza.

Viene mostrato un altro esempio nell'uso di codice letterale specifico per il tipo di composizione. L'intenzione è quella di mostrare un'espressione matematica molto semplice: $123 + 10^{-1}$.

```
<p><tex><![CDATA[$123+10^{-1}$]]></tex>
<html><![CDATA[123+10<sup>-1</sup>]]></html>
<ifinfo>123+10^-1</ifinfo>
= 12.3</p>
```


Si potrebbe notare una sorta di incoerenza nell'uso degli elementi letterali, assieme a '**ifinfo**', il cui scopo è solo quello di essere preso in considerazione quando la composizione produce il formato Info. Il fatto è che gli altri due elementi letterali, oltre che contenere codice letterale per il tipo rispettivo di composizione, sono implicitamente elementi condizionali. Dal momento che la composizione Info non può prevedere una codifica letterale speciale, l'unico modo per integrare le varie parti è quello di usare '**ifinfo**' per rappresentare in qualche modo l'espressione, anche in questo caso.

Problemi

Texinfo, come TeX e *roff, distingue i blocchi di testo in quanto separati da una o più righe vuote. In tal modo, la distinzione tra blocchi di testo e testo interno alle righe, è solo una questione di spazio verticale. Per esempio, il pezzo seguente di un sorgente Texinfo, mostra tre ambienti del tipo '**@if*composizione***', che sono parte dello stesso blocco di testo, ovvero lo stesso paragrafo.

```
La composizione attuale è
@iftex
TeX
@end iftex
@ifhtml
HTML
@end ifhtml
@ifinfo
Info
@end ifinfo
e si può vedere che...
```

In una situazione differente, questi ambienti possono diventare blocchi isolati di testo, come si vede qui:

```
La composizione attuale è:

@iftex
TeX
@end iftex

@ifhtml
HTML
@end ifhtml

@ifinfo
Info
@end ifinfo

Si può vedere che...
```

Con un sistema SGML, questa confusione di ruoli non è desiderabile, oltre che essere difficile da realizzare. Questo è il motivo per cui Sgmltexi distingue tra '**@if*composizione***' o '**@ifnot*composizione***', e '**@if*composizione*block**' o '**@ifnot*composizione*block**'.

Sgmltexi cerca di mantenere le interruzioni di riga contenute all'interno del sorgente SGML, ma per questo ci sono delle conseguenze nell'uso degli ambienti condizionali, del tipo interno alle righe. Ciò dipende dal fatto che necessariamente occorre aggiungere delle interruzioni aggiuntive. Si supponga di voler scrivere qualcosa come ciò che segue:

```
<p>La composizione attuale
è <iftex>TeX</iftex><ifhtml>HTML</ifhtml><ifinfo>Info</ifinfo>, per cui
si sa cosa comporta questo fatto.</p>
```

Ci si aspetta che i marcatori di apertura e di chiusura vengano rimpiazzati aggiungendo anche le interruzioni di riga appropriate. Ma se fosse così, il risultato sarebbe quello seguente, in cui ciò che prima era testo interno alla riga diventa invece un blocco separato:

```
La composizione attuale
è
@iftex
TeX
@end iftex

@ifhtml
HTML
@end ifhtml

@ifinfo
Info
@end ifinfo
, per cui
si sa cosa comporta questo fatto.</p>
```

Per risolvere il problema, questi elementi intesi come ambienti condizionali interni alle righe, non introducono alcuna interruzione iniziale o finale che sia; rimane compito dell'autore il preoccuparsi di questo problema. Per questo, il sorgente di Sgmltexi deve essere scritto come si vede nell'esempio seguente, considerando anche che non c'è alcun modo di mettere la virgola dopo il nome del tipo di composizione.

```
<p>La composizione attuale è
<iftex>TeX</iftex>
<ifhtml>HTML</ifhtml>
<ifinfo>Info</ifinfo>
per cui si sa cosa comporta questo fatto.</p>
```

Lo stesso problema appare con gli elementi '**tex**' e '**html**', ma in tal caso non c'è bisogno di qualificarne il contenuto, che si intende sempre come testo interno alle righe.

```
<p>
<tex>
$$ \chi^2 = \sum_{i=1}^N
          \left (y_i - (a + b x_i)
          \over \sigma_i\right)^2 $$
</tex>
</p>
```

Utilizzando un sistema SGML, l'inserzione di codice letterale per il tipo di composizione particolare che si utilizza, è da considerarsi come l'**ultima risorsa**. In altri termini, se sono necessari tali espedienti, è evidente che l'SGML è la scelta sbagliata per scrivere la propria documentazione.

In questo capitolo conclusivo della parte dedicata a Sgmltexi, si riepiloga brevemente l'uso di questo sistema di composizione, attraverso la comparazione con Texinfo. In questo modo, si può comprendere cosa di Texinfo non è disponibile con Sgmltexi.

Si osservi che nei modelli sintattici, le parentesi graffe hanno significato letterale, facendo parte dei comandi di Texinfo.

@spazio_bianco

```
 
```

@!

```
&esexcl;
```

End sentence exclamation mark

@"*x*

@'*x*

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@*

```
<br>
```

@,{*x*}

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@-

```
<dh>
```

@.

```
&esperiod;
```

End sentence period

@:

```
&nes;
```

Not ending sentence

@=*x*

Non disponibile.

@?

```
&esquest;
```

End of sentence question mark

@@

```
@
```

@^

@'

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@{

```
{
```

@}

```
}
```

@~

@AA{}

@aa{}

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@acronym{*abbreviazione***}**

```
<acronym>abbreviazione</acronym>
```

@AE{}

@ae{}

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@afivepaper

@afourpaper

@afourlatex

@afourwide

In sostituzione di questi comandi, si possono usare le opzioni della riga di comando: '**--paper=a5**', '**--paper=a4**', '**--paper=a4latex**', '**--paper=a4wide**'.

@alias ***nuovo=esistente***

Non disponibile. Probabilmente si può rimediare inserendo il comando all'interno dell'elemento '**texinfo**'.

@anchor{*nome***}**

```
<anchor id="nome">
```

@appendix ***titolo***

@appendixsec ***titolo***

@appendixsection ***titolo***

@appendixsubsec ***titolo***

@appendixsubsection ***titolo***

@appendixsubsubsec ***titolo***

@appendixsubsubsection ***titolo***

Le appendici si ottengono nell'ambito dell'elemento '**appendix**'.

@asis

La parola '**asis**' è usata come argomento dell'attributo '**emphasis**' degli elementi '**table**', '**vtable**' e '**ftable**'.

@author ***autore***

```
<author>autore</author>
```

@b{*testo***}**

```
<bold>testo</bold>
```

@bullet{}

```
&bull;
```


@bye

```
</sgmltexi>
```

@c ***commento***

@comment ***commento***

Non è disponibile un elemento equivalente, dal momento che l'SGML offre un suo sistema per annotare i commenti. Se necessario, questo comando può essere incluso all'interno di un elemento '**texinfo**'.

@cartouche

```
<cartouche>
blocco_di_testo
|
|
</cartouche>
```

@center ***testo***

```
<center>testo</center>
```

Non si può usare nel titolo del documento.

@centerchap ***titolo***

Non disponibile.

@chapheading ***titolo***

```
<h1 type="heading">titolo</h1>
```

@chapter ***titolo***

```
<h1>titolo</h1>
```

@cindex ***voce***

```
<cindex entry="voce">
```

@cite{*riferimento***}**

```
<cite>riferimento</cite>
```

@clear ***indicatore***

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@code{*testo***}**

```
<code>sample</code>
```

@command{*nome_comando***}**

```
<command>nome_comando</command>
```

@contents

```
<contents>
```

@copyright{}

```
&copy;
```

@defcodeindex ***nome_indice***

```
<defcodeindex>nome_indice</defcodeindex>
```

@defcv ***categoria classe nome***

@defcvx ***categoria classe nome***

```
<defcv cat="categoria" class="classe" name="nome">
[<defcvx cat="categoria" class="classe" name="nome">]…
    …
    …
</defcv>
```

@deffn *categoria nome argomento…*

@deffnx *categoria nome argomento…*

```
<deffn cat="categoria" name="nome">
  <args>argomento…</args>
[<deffnx cat="categoria" name="nome">
  <args>argomento…</args>]…
    …
    …
</deffn>
```

@defindex *nome_indice*

```
<defindex>nome_indice</defindex>
```

@definfoenclose *nuovo_comando prima dopo*

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@defivar *classe nome_variabile_di_istanza*

@defivarx *classe nome_variabile_di_istanza*

```
<defivar class="classe" name="nome_variabile_di_istanza">
[<defivarx class="classe" name="nome_variabile_di_istanza">]…
    …
    …
</defivar>
```

@defmac *nome_macro argomento…*

@defmacx *nome_macro argomento…*

```
<defmac name="nome_macro">
  <args>argomento…</args>
[<defmacx name="nome_macro">
  <args>argomento…</args>]…
    …
    …
</defmac>
```

@defmethod *classe nome_metodo argomento…*

@defmethodx *classe nome_metodo argomento…*

```
<defmethod class="classe" name="nome_metodo">
  <args>argomento…</args>
[<defmethod class="classe" name="nome_metodo">
  <args>argomento…</args>]…
    …
    …
</defmethod>
```

@defop *categoria classe nome argomento…*

@defopx *categoria classe nome argomento…*

```
<defop cat="categoria" class="classe" name="nome">
  <args>argomento…</args>
[<defopx cat="categoria" class="classe" name="nome">
  <args>argomento…</args>]…
    …
    …
</defop>
```

@defopt *nome_opzione*

@defoptx *nome_opzione*

```
<defopt name="nome_opzione">
[<defoptx name="nome_opzione">]
    …
    …
</defopt>
```


@defspec *nome argomento…*

@defspecx *nome argomento…*

```
<defspec name="nome">
  <args>argomento…</args>
[<defspecx name="nome">
  <args>argomento…</args>]…
    …
    …
</defspec>
```

@deftp *categoria nome attributo…*

@deftpx *categoria nome attributo…*

```
<deftp cat="categoria" name="nome">
  <args>attributo…</args>
[<deftpx cat="categoria" name="nome">
  <args>attributo…</args>]…
    …
    …
</deftp>
```

@deftypefn *classificazione tipo_dati nome argomento…*

@deftypefnx *classificazione tipo_dati nome argomento…*

```
<deftypefn cat="classificazione" type="tipo_dati" name="nome">
  <args>argomento…</args>
[<deftypefnx cat="classificazione" type="tipo_dati" name="nome">
  <args>argomento…</args>]…
    …
    …
</deftypefn>
```

@deftypefun *tipo_dati nome_funzione argomento…*

@deftypefunx *tipo_dati nome_funzione argomento…*

```
<deftypefun type="tipo_dati" name="nome_funzione">
  <args>argomento…</args>
[<deftypefunx type="tipo_dati" name="nome_funzione">
  <args>argomento…</args>]…
    …
    …
</deftypefun>
```

@deftypeivar *classe tipo_dati nome_variabile*

@deftypeivarx *classe tipo_dati nome_variabile*

```
<deftypeivar class="classe" type="tipo_dati" name="nome_variabile">
[<deftypeivarx class="classe" type="tipo_dati" name="nome_variabile">
]…
    …
    …
</deftypeivar>
```

@deftypemethod *classe tipo_dati nome_metodo argomento…*

@deftypemethodx *classe tipo_dati nome_metodo argomento…*

```
<deftypemethod class="classe" type="tipo_dati" name="nome_metodo">
  <args>argomento…</args>
[<deftypemethodx class="classe" type="tipo_dati" name="nome_metodo">
  <args>argomento…</args>]…
    …
    …
</deftypemethod>
```

@deftypeop *categioria classe tipo_dati nome argomento…*

@deftypeopx *categoria classe tipo_dati nome argomento*…

```
<deftypeop cat="categoria" class="classe" type="tipo_dati" name="nome">
  <args>argomento…</args>
[<deftypeopx cat="categoria" class="classe" type="tipo_dati" name="nome
  <args>argomento…</args>]…
    …
    …
</deftypeop>
```

@deftypevar *tipo_dati nome_variabile*

@deftypevarx *tipo_dati nome_variabile*

```
<deftypevar type="tipo_dati" name="nome_variabile">
[<deftypevarx type="tipo_dati" name="nome_variabile">]…
    …
    …
</deftypeivar>
```

@deftypevr *classificazione tipo_dati nome_variabile*

@deftypevrx *classificazione tipo_dati nome_variabile*

```
<deftypevr class="classificazione" type="tipo_dati" name="nome_variabile">
[<deftypevrx class="classificazione" type="tipo_dati" name="nome_variabile"
]…
    …
    …
</deftypevr>
```

@defun *nome_funzione argomento*…

@defunx *nome_funzione argomento*…

```
<defun name="nome_funzione">
  <args>argomento…</args>
[<defunx name="nome_funzione">
  <args>argomento…</args>]…
    …
    …
</defun>
```

@defvar *nome_variabile*

@defvarx *nome_variabile*

```
<defvar name="nome_variabile">
[<defvarx name="nome_variabile">]…
    …
    …
</defvar>
```

@defvr *categoria nome_variabile*

@defvrx *categoria nome_variabile*

```
<defvr cat="categoria" name="nome_variabile">
[<defvrx cat="categoria" name="nome_variabile">]…
    …
    …
</defvr>
```

@detailmenu

```
<menu>
    …
    …
<detailmenu>
    …
    …
</detailmenu>
</menu>
```


@dfn{*termine***}**

```
<dfn>termine</dfn>
```

@dircategory *dirpart*

@direntry

```
<infodir cat="dirpart">
    …
    …
</infodir>
```

@display

```
<display>
    blocco_di_testo
    …
    …
</display>
```

@dmn{*dimensione***}**

```
<dmn>dimensione</dmn>
```

@documentdescription *descrizione* **@end documentdescription**

```
<documentdescription content="descrizione">
```

@documentencoding *codifica*

```
<sgmltexi charset="codifica">
```

Definisce la codifica del sorgente Texinfo che viene generato, stabilendo implicitamente che lo stesso sorgente SGML è realizzato nello stesso modo. Viene oscurato dall'opzione '**--input-encoding**', che prende la precedenza generando un sorgente Texinfo in formato ISO 646 puro (ASCII a 7 bit).

@documentlanguage *cc*

```
<sgmltexi lang="cc">
```

@dotaccent{*c***}**

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@dots{}

```
&dots;
```

@email{*indirizzo***,** *testo_mostrato***}**

```
<email email="indirizzo" name="testo_mostrato">
```

@emph{*testo***}**

```
<emph>testo</emph>
```

@env{*variabile_di_ambiente***}**

```
<env>variabile_di_ambiente</env>
```

@enddots{}

```
&enddots;
```

@enumerate [*numero_o_lettera*]

```
<enumerate [start="numero_o_lettera"]>
<item>
    …
    …
<item>
    …
    …
</enumerate>
```

@equiv{}

```
&equiv;
```

@error{}

```
&error;
```

@evenfooting

@evenheading

@everyfooting

@everyheading

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@example

```
<example>
    bloco_di_testo
    …
    …
</example>
```

Preformattato:

```
<example>
<pre>
    riga_di_testo
    …
    …
</pre>
</example>
```

Letterale:

```
<example>
<pre>
<![CDATA[
    riga_di_testo
    …
    …
]]>
</pre>
</example>
```

@exampleindent

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@exlamdown

```
&iexcl;
```

@exdent

```
<pre>
    …
<exdent>testo_sporgente</exdent>
    …
</pre>
```


@expansion{}

```
&expansion;
```

@file{*nome_file***}**

```
<file>nome_file</file>
```

@finalout

Non gestibile, in quanto il sorgente Texinfo che viene generato contiene sempre questo comando.

@findex *voce*

```
<findex entry="voce">
```

@flushleft

```
<flushleft>testo</flushleft>
```

@flushright

```
<flushright>testo</flushright>
```

@footnote{*testo_del_piè_pagina***}**

```
<footnote>testo_del_piè_pagina</footnote>
```

@footnotestyle *stile*

```
<footnotestyle content="stile">
```

In alternativa si può usare l'opzione '**--footnotestyle=***stile*' della riga di comando, la quale prevale.

@format

```
<format>
<pre>
    …
    …
</pre>
</format>
```

Letterale:

```
<format>
<pre>
<![CDATA[
    …
    …
]]>
</pre>
</format>
```

@frenchspacing

```
<sgmltexi spacing="french">
```

@ftable *comando_di_composizione*

```
<ftable emphasis="commando">
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
    …
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
</ftable>
```

@group

```
<group>blocco_di_testo</group>
```

@H{*c*}

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@heading ***titolo***

```
<h2 type="heading">titolo</h2>
```

@headings on

@headings off

@headings single

@headings double

```
<headings content="on">
<headings content="off">
<headings content="single">
<headings content="double">
```

In alternativa si può usare l'opzione '**--headings**' della riga di comando, la quale prevale su queste direttive:

```
--headings=on
--headings=off
--headings=single
--headings=double
```

@html

```
<html>codice_html</html>
```

@hyphenation{*parole_separate_in_sillabe***}**

```
<hyphenation words="parole_separate_in_sillabe">
```

@i{*testo***}**

```
<italic>testo</italic>
```

@ifclear ***opzione***

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@ifhtml

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifhtml>testo_interno_alle_righe</ifhtml>
```

```
<ifhtmlblock>
    blocco_di_testo
    …
</ifhtmlblock>
```

L'SGML dà la possibilità di usare le sezioni marcate. Queste posso essere controllate da Sgmltexi attraverso l'opzione '**--sgml-include**' della riga di comando. Per esempio, il sorgente SGML potrebbe essere simile al pezzo seguente:


```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN"
[
<!ENTITY % HTML            "IGNORE">
<!ENTITY % INFO            "IGNORE">
<!ENTITY % TEX             "IGNORE">
…
…
]>
<sgmltexi>
…
…
<![%HT;[
    <p>Here it is some text that is meant to appear only inside
    the HTML typesetting.</p>
]]>
<![%INFO;[
    <p>Here it is some other text that is meant to appear only
    inside the Info typesetting.</p>
]]>
<![%TEX;[
    <p>This text is meant to appear only inside the TeX
    typesetting.</p>
]]>
…
…
</sgmltexi>
```

Quindi, quando si genera la composizione in HTML, si deve utilizzare l'opzione '**--sgml-include=HTML**':

```
$ sgmltexi --sgml-include=HTML --html mio_file.sgml[Invio]
```

Per la composizione nel formato Info, si deve usare l'opzione '**--sgml-include=INFO**':

```
$ sgmltexi --sgml-include=INFO --info mio_file.sgml[Invio]
```

Nello stesso modo, per la composizione attraverso TeX si deve usare l'opzione '**--sgml-include=TEX**':

```
$ sgmltexi --sgml-include=TEX --tex mio_file.sgml[Invio]
```

@ifinfo

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifinfo>testo_interno_alle_righe</ifinfo>
```

```
<ifinfoblock>
    blocco_di_testo
    …
</ifinfoblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**@ifhtml**'.

@ifnothtml

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifnothtml>testo_interno_alle_righe</ifnothtml>
```

```
<ifnothtmlblock>
    blocco_di_testo
    …
</ifnothtmlblock>
```

L'SGML dà la possibilità di usare le sezioni marcate. Queste posso essere controllate da Sgmltexi attraverso l'opzione '**--sgml-include**' della riga di comando. Per esempio, il sorgente SGML potrebbe essere simile al pezzo seguente:

```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN"
[
<!ENTITY % NOTHTML          "IGNORE">
<!ENTITY % NOTINFO          "IGNORE">
<!ENTITY % NOTTEX           "IGNORE">
…
…
]>
<sgmltexi>
…
…
<![%NOTHTML;[
    <p>Here it is some text that is meant to appear only outside
    the HTML typesetting.</p>
]]>
<![%NOTINFO;[
    <p>Here it is some other text that is meant to appear only
    outside the Info typesetting.</p>
]]>
<![%NOTTEX;[
    <p>This text is meant to appear only outside the TeX
    typesetting.</p>
]]>
…
…
</sgmltexi>
```

Quindi, quando si genera la composizione in HTML, si devono utilizzare le opzioni '**`--sgml-include=NOTINFO`**' e '**`--sgml-include=NOTTEX`**':

```
$ sgmltexi --sgml-include=NOTINFO --sgml-include=NOTTEX ↩
↪--html mio_file.sgml[Invio]
```

Per la composizione nel formato Info, si devono utilizzare le opzioni '**`--sgml-include=NOTHTML`**' e '**`--sgml-include=NOTTEX`**':

```
$ sgmltexi --sgml-include=NOTHTML --sgml-include=NOTTEX ↩
↪--info mio_file.sgml[Invio]
```

Nello stesso modo, per la composizione attraverso TeX si devono utilizzare le opzioni '**`--sgml-include=NOTHTML`**' e '**`--sgml-include=NOTINFO`**':

```
$ sgmltexi --sgml-include=NOTHTML --sgml-include=NOTINFO ↩
↪--tex mio_file.sgml[Invio]
```

`@ifnotinfo`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifnotinfo>testo_interno_alle_righe</ifnotinfo>
```

```
<ifnotinfoblock>
    blocco_di_testo
    …
</ifnotinfoblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifnothtml`**'.

`@ifnotplaintext`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifnotplaintext>testo_interno_alle_righe</ifnotplaintext>
```

```
<ifnotplaintextblock>
    blocco_di_testo
    …
</ifnotplaintextblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifnothtml`**'.

`@ifnottex`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifnottex>testo_interno_alle_righe</ifnottex>
```

```
<ifnottexblock>
    blocco_di_testo
    …
</ifnottexblock>
```


L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifnothtml`**'.

`@ifnotxml`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifnotxml>testo_interno_alle_righe</ifnotxml>
```

```
<ifnotxmlblock>
    blocco_di_testo
    …
</ifnotxmlblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifnothtml`**'.

`@ifplaintext`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifplaintext>testo_interno_alle_righe</ifplaintext>
```

```
<ifplaintextblock>
    blocco_di_testo
    …
</ifplaintextblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifnothtml`**'.

`@ifset` *flag*

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**`texinfo`**'.

`@iftex`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<iftex>testo_interno_alle_righe</iftex>
```

```
<iftexblock>
    blocco_di_testo
    …
</iftexblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifhtml`**'.

`@ifxml`

Ci sono due possibilità: testo interno alle righe e blocchi di testo.

```
<ifxml>testo_interno_alle_righe</ifxml>
```

```
<ifxmlblock>
    blocco_di_testo
    …
</ifxmlblock>
```

L'SGML dà la possibilità di usare le sezioni marcate, come è già stato mostrato a proposito del comando '**`@ifhtml`**'.

`@ignore`

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**`texinfo`**'. Se non è necessario inserire commenti nel file Texinfo che viene generato, si possono usare i commenti secondo l'SGML:

```
<!--
    commento
    …
    …
-->
```

@image{*nome_file*, [*ampiezza*], [*altezza*]}

```
<image name="nome_file" width="ampiezza" height="altezza">
```

@include

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'. L'SGML offre un meccanismo alternativo:

```
<!DOCTYPE Sgmltexi PUBLIC "-//GNU//DTD Sgmltexi//EN"
[
<!ENTITY GPL        SYSTEM "licenses/gpl.sgml">
<!ENTITY BSD        SYSTEM "licenses/bsd.sgml">
…
…
]>
<sgmltexi>
…
…
<appendix>
&GPL;
&BSD;
…
…
</appendix>
…
</sgmltexi>
```

come si può vedere dall'esempio, l'inserzione nel testo di '`licenses/gpl.sgml`' e di '`licenses/bsd.sgml`' avviene attraverso l'uso delle macro SGML '**&GPL;**' e '**&BSD;**'.

Se è necessario includere un file Texinfo, si può fare come si vede nell'esempio seguente:

```
<![CDATA[
<p><texinfo>
@include example.texi
</texinfo></p>
]]>
```

È necessario tenere a mente che l'elemento '**texinfo**' è di tipo interno alle righe di testo. Ecco perché nell'esempio è contenuto in un elemento '**p**'.

@inforef{*nome_nodo*, [*voce*], *nome_file_info*}

```
<inforef id="nome_nodo" name="voce" info="nome_file_info">
```

\input ***file_macro***

Non è possibile inserire macro aggiuntive all'inizio del documento, oltre a quella predefinita che imposta la sintassi Texinfo.

@item

Questo comando di Texinfo viene usato in contesti molto diversi. All'interno di Sgmltexi non esiste un modo unico per utilizzarlo, per cui conviene vedere piuttosto la descrizione dei comandi '**@table**', '**@ftable**', '**@vtable**', '**@itemize**', '**@enumerate**' e '**@multitable**'.

@itemize [*marcatore_iniziale*]

```
<itemize [mark="marcatore_iniziale"]>
<item>
    …
    …
<item>
    …
    …
</itemize>
```

@itemx

Questo comando di Texinfo viene usato in contesti molto diversi. All'interno di Sgmltexi non esiste un modo unico per utilizzarlo, per cui conviene vedere piuttosto la descrizione dei comandi '**@table**', '**@ftable**' e '**@vtable**'.

@kbd{*tasti_premuti*}

```
<kbd>tasti_premuti</kbd>
```


@kbdinputstyle ***stile***

```
<kbdstyle style="stile">
```

@key{*nome_tasto*}

```
<key>nome_tasto</key>
```

@kindex ***voce***

```
<kindex entry="voce">
```

@L{}

```
&Lstrok;
```

@l{}

```
&lstrok;
```

@lisp

```
<lisp>
    blocco_di_testo
    …
    …
</lisp>
```

Preformattato:

```
<lisp>
<pre>
    riga_di_testo
    …
    …
</pre>
</lisp>
```

Letterale:

```
<lisp>
<pre>
<![CDATA[
    riga_di_testo
    …
    …
]]>
</pre>
</lisp>
```

@lowersections

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@macro ***nome_macro {parametri}***

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@majorheading ***titolo***

Non disponibile attualmente.

@math{*espressione_matematica*}

```
<math>espressione_matematica</math>
```

@menu

```
<menu>[menù_info</menu>]
```

@minus{}

```
&minus;
```

@multitable *larghezza_delle_colonne*

```
<multitable>
<columnfraction>frazione_larghezza_complessiva</columnfraction>…
<raw>cella[<tab>cella]…</raw>…
…
</multitable>
```

```
<multitable>
<columnexample>testo_di_esempio</columnexample>…
<raw>cella[<tab>cella]…</raw>…
…
</multitable>
```

@need *n*

```
<need mils="n">
```

@node *nome*, *successivo*, *precedente*, *superiore*

La gestione manuale dei nodi di Texinfo avviene come si vede nello schema seguente, dove ci si limita a stabilire il nome del nodo in questione:

```
<hn node="nome">titolo</hn>
```

Se è necessario un controllo completo sui nodi, si possono stabilire anche gli altri dati, come nello schema seguente:

```
<hn node="nome" next="successivo" prev="precedente" up="superiore">titolo<
```

Sgmltexi non fa alcun controllo di validità per quanto riguarda l'inserzione manuale dei nodi.

@noindent

```
<p indent="off">
```

@novalidate

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

O{}

```
&Oslash;
```

o{}

```
&oslash;
```

@oddfooting

@oddheading

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@option{*opzione***}**

```
<option>opzione</option>
```

@page

```
<page>
```

@pagesizes [*ampiezza*][, *altezza*]

Non disponibile.

@paragraphindent *rientro*

Non disponibile.

@pindex *voce*

```
<pindex entry="voce">
```


@point{}

```
&point;
```

@pounds{}

```
&pound;
```

@print{}

```
&print;
```

@printindex *nome_indice*

```
<printindex name="nome_indice">
```

@pxref{*nome_nodo*, [*voce*], [*argomento_o_titolo*], [*file_info*], [*manual*]**}**

```
<pxref id="nome_nodo" name="voce" title="argomento_o_titolo" info="file_
  ptitle="manual">
```

@questiondown{}

```
&iquest;
```

@quotation

```
<quotation>
    testo_interno_alle_righe
    …
    …
</quotation>
```

@r{*testo***}**

```
<roman>testo</roman>
```

@raisesections

Non disponibile.

@ref{*nome_nodo*, [*voce*], [*argomento_o_titolo*], [*file_info*], [*manuale*]**}**

```
<ref id="nome_nodo" name="voce" title="argomento_o_titolo" info="file_inf
  ptitle="manuale">
```

@refill

Non disponibile.

@result{}

```
&result;
```

@ringaccent{*c***}**

Per la rappresentazione di caratteri speciali, si possono utilizzare le entità standard SGML, oppure i caratteri della codifica ISO 8859-***n*** selezionata con l'opzione '**--input-encoding**', o con l'attributo '**charset**' dell'elemento '**sgmltexi**'.

@samp{*testo***}**

```
<samp>testo</samp>
```

@sc{*testo***}**

```
<sc>testo</sc>
```

@section *titolo*

```
<h2>titolo</h2>
```

@set *flag string*

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

`@setchapternewpage on`

`@setchapternewpage off`

`@setchapternewpage odd`

Si può usare l'elemento '**`setchapternewpage`**', come negli schemi seguenti:

```
<setchapternewpage content="on">
<setchapternewpage content="off">
<setchapternewpage content="odd">
```

In alternativa si può utilizzare l'opzione '**`--setchapternewpage`**', nella riga di comando:

```
--setchapternewpage=on
--setchapternewpage=off
--setchapternewpage=odd
```

`@setcontentsaftertitlepage`

Non disponibile.

`@setfilename` ***nome_file_info***

```
<setfilename content="nome_file_info">
```

`@setshortcontentsaftertitlepage`

Non disponibile.

`@settitle` ***titolo***

```
<settitle content="titolo">
```

`@shortcontents`

```
<shortcontents>
```

`@shorttitlepage` ***title***

Non disponibile.

`@smallbook`

Si usa per questo l'opzione: '**`--paper=small`**'.

`@smalldisplay`

```
<smalldisplay>
    blocco_di_testo
    …
    …
</smalldisplay>
```

`@smallexample`

```
<smallexample>
    bloco_di_testo
    …
    …
</smallexample>
```

Preformattato:

```
<smallexample>
<pre>
    riga_di_testo
    …
    …
</pre>
</smallexample>
```

Letterale:

```
<smallexample>
<pre>
<![CDATA[
    riga_di_testo
    …
    …
]]>
</pre>
</smallexample>
```

`@smallformat`

```
<smallformat>
<pre>
    …
    …
</pre>
</smallformat>
```

Letterale:

```
<smallformat>
<pre>
<![CDATA[
    …
    …
]]>
</pre>
</smallformat>
```

`@smalllisp`

```
<smalllisp>
    blocco_di_testo
    …
    …
</smalllisp>
```

Preformattato:

```
<smalllisp>
<pre>
    riga_di_testo
    …
    …
</pre>
</smalllisp>
```

Letterale:

```
<smalllisp>
<pre>
<![CDATA[
    riga_di_testo
    …
    …
]]>
</pre>
</smalllisp>
```

`@sp` ***n***

```
<sp lines="n">
```

`@ss{}`

```
&szlig;
```

`@strong{`*testo***`}`**

```
<strong>testo</strong>
```

@subheading *titolo*

```
<h3 type="heading">titolo</h3>
```

@subsection *titolo*

```
<h3>titolo</h3>
```

@subsubheading *titolo*

```
<h4 type="heading">titolo</h4>
```

@subsubsection *titolo*

```
<h4>titolo</h4>
```

@subtitle *sottotitolo*

```
<subtitle>sottotitolo</subtitle>
```

@summarycontents

```
<summarycontents>
```

@syncodeindex *indice_di_origine indice_di_destinazione*

```
<syncodeindex from="indice_di_origine" to="indice_di_destinazione">
```

@synindex *indice_di_origine indice_di_destinazione*

```
<synindex from="indice_di_origine" to="indice_di_destinazione">
```

@t{*testo***}**

```
<typewriter>testo</typewriter>
```

@tab

Si veda la descrizione di '**@multitable**'.

@table *comando_di_composizione*

```
<table emphasis="commando">
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
    …
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
    …
</table>
```

@TeX{}

```
&TeX;
```

@tex

```
<tex>pezzo_di_sorgente_tex</tex>
```

@thischapter

@thischaptername

@thisfile

@thispage

@thistitle

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@tie{}

```
 
```


@tieaccent{*cc***}**

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@tindex *voce*

```
<tindex entry="voce">
```

@title *titolo*

```
<title>titolo</title>
```

@titlefont{*testo***}**

Non disponibile.

@titlepage

Non disponibile. Si veda come è organizzata la struttura di Sgmltexi.

@today

```
&today;
```

@top

Viene generato automaticamente.

@u{*c***}**

@ubaraccent{*c***}**

@udotaccent{*c***}**

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@unnumbered *titolo*

```
<h1 type="unnumbered">titolo</h1>
```

@unnumberedsec *titolo*

```
<h2 type="unnumbered">titolo</h2>
```

@unnumberedsubsec *titolo*

```
<h3 type="unnumbered">titolo</h3>
```

@unnumberedsubsubsec *titolo*

```
<h4 type="unnumbered">titolo</h4>
```

@uref{*url***, [***testo_mostrato***], [***rimpiazzo***]}**

```
<uref uri="url" name="testo_mostrato" replace="rimpiazzo">
```

@url{*url***}**

```
<url>url</url>
```

@v{*c***}**

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@value{*indicatore***}**

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@var{*metavariabile***}**

```
<var>metavariabile</var>
```

@verb{*x testo_letterale x***}**

```
<verb char="x"><[CDATA[testo_letterale]]></verb>
```

@verbatim *testo_letterale* **@end verbatim**

```
<verbatim>
<[CDATA[
testo_letterale
]]>
</verbatim>
```

@verbatiminclude *file*

```
<verbatiminclude file="file">
```

@vindex *voce*

```
<vindex entry="voce">
```

@vskip *ammontare_dell'avanzamento*

Non disponibile. Eventualmente può essere usato all'interno dell'elemento '**texinfo**'.

@vtable *comando_di_composizione*

```
<vtable emphasis="commando">
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
    …
<item>voce_descrittiva</item>
[<itemx>voce_descrittiva</itemx>]…
    blocco_di_testo…
    …
    …
</vtable>
```

@w{*testo***}**

```
<whole>testo</whole>
```

@xref{*nome_nodo***,** [*voce*]**,** [*argomento_o_titolo*]**,** [*file_info*] **,** [*manuale*]**}**

```
<xref id="nome_nodo" name="voce" title="titolo_o_argomento" info="file_in
  ptitle="manuale">
```


Sistemi vari di composizione elettronica

Introduzione a Lout 677
- Collocazione dei componenti di Lout 677
- Funzionamento 678
- Esempio introduttivo 678
- Concetti fondamentali di Lout 680
- Caratteri speciali e stringhe letterali 681
- Spazi e spaziature 682
- Elementi essenziali di un documento Lout 682
- Argomenti dei comandi e unità di misura 689
- Rappresentazione simbolica della codifica 689
- Caratteri da stampa 691
- Display 692
- Caratteristiche interne dei paragrafi 693
- Testo letterale 695
- Elenchi 695
- Note 697
- Figure e tabelle fluttuanti 698
- Indici e riferimenti incrociati 700
- Localizzazione 702
- Personalizzazione dello stile 703
- Riferimenti 704

Introduzione a LyX 705
- Creazione di un documento 705
- Struttura e stile 706
- Modelli di documento 707
- Automatismi 707
- Riferimenti 707

Introduzione a HieroTeX 709
- Installazione 709
- Utilizzare HieroTeX 710
- Codifica di HieroTeX 714
- Riferimenti 720

Trasformazione in altri formati 721
- DLH: trasforma LaTeX in HTML 721
- Help2man: genera una pagina di manuale dalle informazioni fornite dal programma 722
- Pstotext: estrae il testo da un file PostScript o PDF 723
- Mswordview 723
- Catdoc 723
- Antiword 724
- xlHTML 725

Collocazione dei componenti di Lout 677
Funzionamento 678
Esempio introduttivo 678
Concetti fondamentali di Lout 680
Caratteri speciali e stringhe letterali 681
Spazi e spaziature 682
Elementi essenziali di un documento Lout 682
Dichiarazione dello stile generale 682
Preambolo 683
Struttura del documento ordinario 684
Struttura della relazione tecnica 685
Struttura del libro 685
Struttura dei lucidi per lavagna luminosa 686
Sottostrutture 686
Suddivisione del testo 688
Argomenti dei comandi e unità di misura 689
Rappresentazione simbolica della codifica 689
Caratteri da stampa 691
Corpo 691
Display 692
Caratteristiche interne dei paragrafi 693
Interruzione e allineamento 693
Distanza tra le righe 693
Separazione in sillabe 694
Raccogliere tutto assieme 694
Testo letterale 695
Elenchi 695
Note 697
Note a piè pagina e note finali 697
Note a margine 697
Figure e tabelle fluttuanti 698
Figure 699
Tabelle 699
Indici e riferimenti incrociati 700
Riferimenti nel testo 700
Indice generale e indice analitico 701
Problemi connessi alla generazione dei riferimenti incrociati 702
Localizzazione 702
Configurazione di una localizzazione 703
Personalizzazione dello stile 703
Inclusione di altri stili 703
Veste grafica del documento 704
Particolarità del tipo di documento 704
Parte conclusiva 704
Riferimenti 704

Lout [1] è un sistema di editoria elettronica relativamente recente, che deriva dall'esperienza di *roff e TeX, le cui potenzialità sono comparabili con quelle di TeX/LaTeX.

http://lout.sourceforge.net

Collocazione dei componenti di Lout

Lout non è strutturato in una miriade di directory come succede alle distribuzioni LaTeX; è comunque necessario sapere dove sono state collocate alcune sue componenti. Per scoprirlo basta usare il comando seguente:

```
$ lout -V [Invio]
```

Con questo si potrebbe ottenere un messaggio simile a quello seguente:

```
Basser Lout Version 3.08 (May 1996)
Basser Lout written by:      Jeffrey H. Kingston (jeff@cs.usyd.edu.au)
Free source available from:  ftp://ftp.cs.usyd.edu.au/jeff/lout
This executable compiled:    11:30:04 Aug 16 1998
System include directory:    /usr/lib/lout/include
System database directory:   /usr/lib/lout/data
Database index files created afresh automatically: yes
```

L'utente comune potrebbe non avere alcun bisogno di accedere a queste directory; comunque se si vuole realizzare un proprio stile personalizzato, occorre sapere che i file standard sono contenuti nella directory '`/usr/lib/lout/include/`' (in questo caso), essendo la directory delle inclusioni di sistema (come la definisce Lout).

Quando per qualche motivo si interviene nei file di configurazione di Lout contenuti in queste directory, è necessario sapere che poi Lout ha bisogno di generare dei file paralleli (per esempio da '`/usr/lib/lout/data/standard.ld`' viene generato '`/usr/lib/lout/data/standard.li`'). Lout fa le cose in modo automatico appena si accorge della necessità, tuttavia può darsi che in quel momento non abbia i permessi necessari per modificare o creare questi file. Bisogna tenere conto di questa possibilità, provvedendo a sistemare temporaneamente i permessi se ciò accade.

Funzionamento

Allo stato attuale, Lout legge un sorgente e genera un risultato finale in PostScript.

Di solito si avvia l'eseguibile '**`lout`**' senza opzioni, con l'unico argomento costituito dal nome del file sorgente da convertire, in pratica secondo lo schema seguente:

```
lout file_sorgente > file_PostScript
```

Nonostante la ridirezione dello standard output, Lout emette altri messaggi attraverso lo standard error, meno dettagliati di quanto faccia TeX, ma altrettanto importanti. Una cosa da notare subito di Lout è che potrebbe essere necessario ripetere l'operazione di composizione più volte per poter risolvere i riferimenti incrociati, anche in presenza di documenti molto banali.

Esempio introduttivo

La documentazione originale, scritta dallo stesso autore di Lout, parte da esempi molto semplificati per spiegare il comportamento di questo sistema di composizione; tuttavia, le possibilità del linguaggio di Lout potrebbero confondere; pertanto si preferisce mostrare qui un esempio un po' più complesso di quanto si veda di solito, ma allineato al genere di esempi già presentati per gli altri sistemi di composizione.

```
# Sorgente Lout di esempio. Per ottenere il risultato finale
# basta usare il comando:
# lout FILE_LOUT > FILE_PS

@SysInclude { doc }
@Document
    @InitialFont { Times Base 24p }
#   @InitialBreak { adjust 1.2fx hyphen }
#   @InitialSpace { lout }
#   @InitialLanguage{ English }
#   @PageHeaders { Simple }
#   @FirstPageNumber { 1 }
#   @ColumnNumber { 1 }
#   @OptimizePages { No }
//
@Text @Begin

@Display @Heading { Introduzione a Lout }

Questo è un esempio di documento scritto con
Lout. Come si può vedere è stato definito
uno stile generale: doc.

@BeginSections
@Section
    @Title { Suddivisione del documento }
@Begin

@PP
Lo stile «doc» permette una suddivisione del
testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni
@End @Section

@Section
    @Title { Paragrafi }
@Begin

@PP
Il testo di un paragrafo inizia generalmente dopo il
simbolo "@PP", mentre non è presente la possibilità
di staccare i paragrafi solo attraverso una riga
vuota, come accade con TeX. Di solito, se non è
stato cambiato lo stile standard, la prima riga
appare rientrata.

@PP
Attenzione: gli spazi orizzontali,           vengono
rispettati!
@End @Section

@EndSections
@End @Text
```

È fondamentale per Lout che l'ultima riga utile del sorgente sia terminata da un'interruzione di riga. Se ci sono più righe vuote alla fine del sorgente, queste non creano problemi in ogni caso.

Supponendo di abbinare a questo file il nome '`esempio`', si può utilizzare il comando seguente per comporre il documento e ottenere un file PostScript.

```
$ lout esempio > esempio.ps [Invio]
```

Per quanto strano possa sembrare, la prima volta vengono segnalati una serie di avvertimenti su dei riferimenti incrociati non risolti.

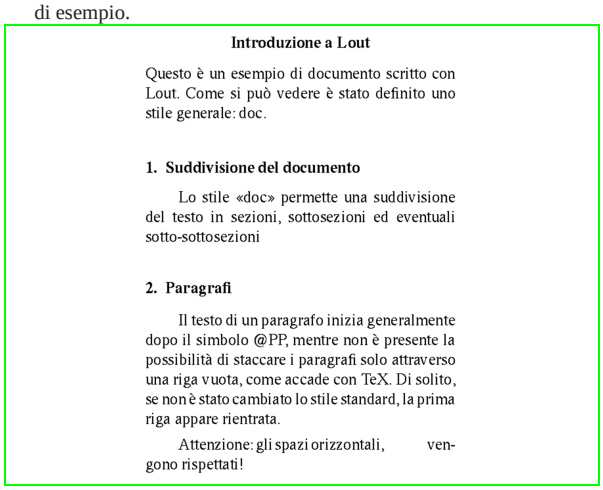
```
lout file "esempio":
    25,1: unresolved cross reference @SectionList&&357.esempio.1
    25,1: unresolved cross reference @SectionList&&357.esempio.1
    35,1: unresolved cross reference @SectionList&&357.esempio.2
    35,1: unresolved cross reference @SectionList&&357.esempio.2
```

Nel frattempo Lout ha creato alcuni file attorno a '`esempio`': '`lout.li`' e '`esempio.ld`' (viene creato anche '`esempio.ps`', ma non si tratta dell'edizione completa). La presenza di questi serve successivamente a risolvere parte o tutti i riferimenti incrociati.

```
$ lout esempio > esempio.ps [Invio]
```

In questo caso, la seconda volta che viene eseguito il comando si ottiene il risultato finale corretto.

Figura u93.4. Il risultato della composizione del sorgente Lout di esempio.



Introduzione a Lout

Questo è un esempio di documento scritto con Lout. Come si può vedere è stato definito uno stile generale: doc.

1. Suddivisione del documento

Lo stile «doc» permette una suddivisione del testo in sezioni, sottosezioni ed eventuali sotto-sottosezioni

2. Paragrafi

Il testo di un paragrafo inizia generalmente dopo il simbolo @PP, mentre non è presente la possibilità di staccare i paragrafi solo attraverso una riga vuota, come accade con TeX. Di solito, se non è stato cambiato lo stile standard, la prima riga appare rientrata.

Attenzione: gli spazi orizzontali, vengono rispettati!

Quando si modifica un documento dopo averlo già elaborato una volta con Lout, potrebbe essere opportuno eliminare i file generati in fase di composizione, in quanto questi possono produrre segnalazioni di errore fasulle, o comunque portare a un risultato finale errato.

Tra la documentazione che accompagna Lout si possono trovare i manuali di questo sistema di composizione, di solito anche in forma sorgente (sono scritti ovviamente in Lout). Questi possono essere ricompilati per ottenere un file PostScript e ciò permette di vedere cosa sia necessario fare di fronte a documenti più complessi. Per la precisione si tratta di documenti articolati in più sorgenti distinti, aggregati globalmente dal file '`all`' (viene usato lo stesso nome per ogni manuale). Si intuisce che il comando di composizione debba essere simile a quello seguente (se non si dispone dei permessi di scrittura nella directory in cui si interviene, forse conviene lavorare su una copia), ma si può osservare che prima di riuscire a ottenere un risultato finale corretto, occorre riavviare il comando più volte, fino a quando non ci sono più riferimenti incrociati da risolvere:

```
$ lout all > risultato.ps [Invio]
```

Concetti fondamentali di Lout

I comandi di Lout sono composti da ***simboli***, ovvero delle parole chiave, che possono essere precedute e seguite da degli argomenti opzionali, il cui scopo è generalmente quello di intervengono su un ***oggetto*** posto alla loro destra (dopo le opzioni eventuali). È difficile esprimere il concetto in modo astratto, ma ancora più difficile è mostrarne un modello sintattico. All'interno di questi comandi vengono usate spesso le parentesi graffe, per raggruppare una serie di oggetti o una serie di argomenti; per questa ragione, nei modelli sintattici (semplificativi) che vengono mostrati, le parentesi graffe vanno intese in senso letterale, come facenti parte del comando.

Le parole chiave con cui sono definiti i simboli sono composte da ***lettere***, che per Lout sono le lettere alfabetiche normali, maiuscole e minuscole (eventualmente anche accentate, ma in generale questo è meglio evitarlo), il carattere '`@`' e il trattino basso ('`_`'). In generale, i simboli più comuni iniziano con il carattere '`@`', in modo che la parola chiave che si ottiene non possa essere confusa con il testo normale, ma esistono comunque dei simboli che non rispettano questa consuetudine e di conseguenza vanno usati solo in contesti particolari. Il fatto che per Lout il carattere '`@`' valga come una lettera normale, fa sì che possano esistere dei simboli (cioè delle parole chiave) che lo contengono all'interno; questo serve a capire che due parole chiave non possono essere aderenti, ma vanno spaziate in modo da consentire la loro individuazione.

Lout basa la sua filosofia su degli oggetti tipografici. Per comprenderlo si osservi l'esempio seguente.

```
Ecco qui: @I ciao a tutti. Sì, proprio @I { a tutti }.
```

Semplificando il concetto, un oggetto è una parola, compresa la punteggiatura che dovesse risultare attaccata a questa, oppure un raggruppamento di oggetti che si ottiene delimitandoli tra parentesi graffe. Osservando l'esempio, il simbolo '`@I`' serve a ottenere il corsivo dell'oggetto che segue, dopo uno o più spazi che per i fini della composizione vengono ignorati. Pertanto, la prima volta che appare '`@I`', questo serve a rendere in corsivo la parola '`ciao`' che appare subito dopo, mentre la seconda, essendoci le parentesi graffe, il corsivo riguarda le parole '`a tutti`'. Si osservi che lo spazio contenuto tra le parentesi graffe, prima della parola '`a`' e dopo la parola '`tutti`', viene semplicemente ignorato ai fini della composizione tipografica. Si osservi ancora che il punto è stato lasciato fuori dal raggruppamento proprio per evitare che venga coinvolto dalla trasformazione in corsivo.

Da questo si può intendere che le parentesi graffe non servono solo a raggruppare degli oggetti, ma anche a dividere ciò che altrimenti sarebbe interpretato come un oggetto unico.

A fianco dell'uso delle parentesi graffe per delimitare un oggetto (raggruppando o dividendo degli oggetti preesistenti) si aggiungono le stringhe letterali, che sono a loro volta degli oggetti interpretati in modo letterale da Lout. Per esempio,

```
Il comando "@I ciao" genera il corsivo della parola «ciao».
```

si traduce in pratica nel testo seguente,

```
Il comando @I ciao genera il corsivo della parola «ciao».
```

dove si riesce a riportare nel risultato finale anche la lettera '`@`', che altrimenti verrebbe assorbita per generare il corsivo.

Le stringhe vanno usate con parsimonia, perché generano degli oggetti che non possono essere suddivisi su più righe, comunque sono l'unico mezzo per rappresentare alcuni simboli che Lout altrimenti interpreterebbe.

Nell'esempio introduttivo si può notare l'uso del carattere '`#`' che introduce un commento fino alla fine della riga. In pratica, questo serve a fare ignorare al sistema di composizione il testo che segue tale simbolo. Spesso, come è stato fatto nell'esempio, si commentano delle istruzioni di Lout che rappresentano un comportamento predefinito, per ricordare il punto in cui andrebbero collocate se fosse necessario cambiarne l'impostazione.

Caratteri speciali e stringhe letterali

Fino a questo punto dovrebbe essere chiaro che le parentesi graffe, il carattere '`@`', il carattere '`#`' e gli apici doppi sono simboli che hanno un significato speciale, o possono essere interpretati in modo particolare. Oltre a questi se ne aggiungono altri e per tutti si pone il problema di poterli inserire nel testo in modo letterale, quando necessario. Ciò si ottiene con le stringhe letterali, delimitate tra apici doppi, come in parte è già stato notato. Usando le stringhe letterali resta comunque la difficoltà di rappresentare gli apici doppi, che così si ottengono con un carattere di escape aggiuntivo: la barra obliqua inversa. Questa, può essere usata **solo** all'interno delle stringhe letterali per mantenere invariato il significato letterale del carattere che la segue immediatamente; di conseguenza, per rappresentare una barra obliqua inversa, occorre usare una stringa letterale, confermando tale barra con un'altra barra obliqua inversa anteriore. La tabella u93.8 mostra l'elenco dei caratteri speciali per Lout e il modo di ottenerli all'interno delle stringhe letterali.

Tabella u93.8. Caratteri speciali di Lout e modo di ottenerli letteralmente all'interno delle stringhe.

Carattere speciale	Stringa letterale per ottenerlo	Carattere speciale	Stringa letterale per ottenerlo
(spazio)	" "	"	"\""
#	"#"	&	"&"

Carattere speciale	Stringa letterale per ottenerlo	Carattere speciale	Stringa letterale per ottenerlo
/	"/"	@	"@"
\	"\\"	^	"^"
{	"{"	|	"|"
}	"}"	~	"~"

Spazi e spaziature

Lout ha una gestione particolare degli spazi verticali e orizzontali. La prima cosa da notare è che le righe vuote non bastano a separare i paragrafi; per questo si usano comandi specifici, come '**@PP**' per esempio, che serve a introdurre il testo di un paragrafo. Pertanto, le righe vuote (una o più di una) vengono trattate al pari di spazi orizzontali aggiuntivi.

Gli spazi orizzontali normali, comprese le interruzioni di riga che si trasformano in spazi orizzontali, vengono rispettati; in particolare, il carattere di tabulazione viene interpretato come l'inserzione di otto spazi normali.

Questo comportamento predefinito di Lout potrebbe non essere desiderabile, per cui si può controllare attraverso l'opzione '**@InitialSpace**' che riguarda praticamente tutti i tipi di documento previsti da Lout. Il simbolo '**@InitialSpace**' prevede un argomento composto da una parola chiave (racchiusa tra parentesi graffe), che esprime il tipo di comportamento riferito alla gestione degli spazi:

Opzione	Descrizione
`@InitialSpace { lout }`	rappresenta l'impostazione predefinita, come è già stato descritto;
`@InitialSpace { troff }`	richiede un comportamento simile a quello di Troff;
`@InitialSpace { tex }`	richiede un comportamento simile a quello di TeX, in cui una sequenza di due o più spazi si traducono semplicemente in uno solo nel risultato finale.

Elementi essenziali di un documento Lout

Lout, come LaTeX, è un po' delicato per quanto riguarda la sequenza di utilizzo di alcune istruzioni che definiscono la struttura del documento. A volte sono disponibili comandi differenti per fare le stesse cose, per esempio attraverso comandi abbreviati o semplificati. Benché si tratti di un sistema ben ordinato, si rischia di fare confusione. In questo senso, quando è possibile scegliere, qui vengono mostrate le forme più prolisse.

Dichiarazione dello stile generale

Un sorgente Lout inizia generalmente con l'inclusione di un file esterno che serve a definire lo stile generale del documento. Nell'esempio introduttivo, dopo una serie di commenti, viene incluso lo stile '**doc**', attraverso il simbolo '**@SysInclude**'. Il comando '**@SysInclude { doc }**' serve a inserire il contenuto del file '`doc`' che si trova nella directory di inclusione nel sistema di Lout; in questo caso, seguendo quanto visto all'inizio del capitolo, si tratta di '`/usr/lib/lout/include/`'.

I tipi di documento principali che sono stati predisposti dall'autore di Lout sono:

Tipo	Descrizione
`doc`	un documento «ordinario» senza caratteristiche particolari;
`report`	il modello di una relazione tecnica;
`book`	un libro suddiviso in capitoli ed eventualmente in parti;
`slides`	un modello per le diapositive e per i lucidi da usare con la lavagna luminosa.

Preambolo

Dopo l'inclusione dello stile si colloca normalmente un simbolo (di Lout) adatto al tipo di documento. Questo prevede una serie di opzioni e si conclude con due barre oblique. Nell'esempio introduttivo, trattandosi di un documento ordinario, si usava un preambolo simile a quello seguente; in questo caso però, vengono mostrate tutte le opzioni disponibili, indicate secondo il loro valore predefinito.

```
@SysInclude { doc }
@Document
    @InitialFont { Times Base 12p }
    @InitialBreak { adjust 1.2fx hyphen }
    @InitialSpace { lout }
    @InitialLanguage{ English }
    @PageHeaders { Simple }
    @FirstPageNumber { 1 }
    @ColumnNumber { 1 }
    @OptimizePages { No }
//
```

Nell'esempio, dopo l'inclusione dello stile '**doc**', appare il simbolo '**@Document**'; i simboli che si vedono sotto sono le sue opzioni e come tali vengono usati normalmente solo quando necessario per alterare alcune impostazioni predefinite. Può essere conveniente mettere tutte le opzioni disponibili, commentando quelle per le quali non c'è bisogno di alterarne l'impostazione predefinita, esattamente come si è fatto nell'esempio introduttivo. Ciò è utile quando si vuole rimaneggiare il documento senza fare troppa fatica, senza dover cercare le informazioni necessarie.

Il preambolo del documento di tipo '**report**' è quello che si vede nell'esempio seguente:

```
@SysInclude { report }
@Report
    @Title {}
    @Author {}
    @Institution {}
    @DateLine { No }
    @CoverSheet { Yes }
    @InitialFont { Times Base 12p }
    @InitialBreak { hyphen adjust 1.2fx }
    @InitialSpace { lout }
    @InitialLanguage { English }
    @PageHeaders { Simple }
    @ColumnNumber { 1 }
    @FirstPageNumber { 1 }
    @OptimizePages { No }
//
```

Si può osservare che alcuni simboli che descrivono delle opzioni hanno un argomento predefinito costituito da un oggetto nullo: '**{}**'.

Nel seguito vengono mostrati i preamboli del documento di tipo '**book**' e '**slide**'.

```
@SysInclude { book }
@Book
    @Title {}
    @Author {}
    @Edition {}
    @Publisher {}
    @BeforeTitlePage {}
    @AfterTitlePage {}
    @InitialFont { Times Base 12p }
    @InitialBreak { adjust 1.2fx hyphen }
    @InitialSpace { lout }
    @InitialLanguage { English }
    @PageHeaders { Titles }
    @ColumnNumber { 1 }
    @FirstPageNumber { 1 }
    @IntroFirstPageNumber { 1 }
    @OptimizePages { No }
//
```

```
@SysInclude { slides }
@OverheadTransparencies
    @Title {}
    @RunningTitle {}
    @Author {}
    @Institution {}
    @DateLine { No }
    @InitialFont { Times Base 20p }
    @InitialBreak { ragged 1.2fx nohyphen }
    @InitialSpace { lout }
    @InitialLanguage { English }
    @PageHeaders { Titles }
    @FirstPageNumber { 1 }
    @FirstOverheadNumber { 1 }
    @FirstLectureNumber { 1 }
    @OptimizePages { No }
//
```

Osservando gli esempi mostrati, si possono notare quali siano le opzioni più frequenti. Vale la pena di accennare subito ad alcune di queste.

Opzione	Descrizione
`@Title`	Come si può immaginare, il simbolo '**`@Title`**' serve come opzione per definire il titolo del documento; si può usare in tutte le situazioni in cui ciò possa avere senso. Infatti, nel caso del documento ordinario non è prevista questa possibilità.
`@InitialFont`	Il simbolo '**`@InitialFont`**' serve a definire il tipo di carattere e il corpo da utilizzare nel testo normale.
`@InitialBreak`	Il simbolo '**`@InitialBreak`**' serve a definire le caratteristiche dei paragrafi di testo normali; in particolare l'allineamento, la distanza tra le righe e l'attivazione o meno della separazione in sillabe delle parole.
`@InitialSpace`	Come già descritto in precedenza, il simbolo '**`@InitialSpace`**' serve a definire il comportamento di Lout nei confronti degli spazi, nel senso di stabilire se questi devono essere rispettati oppure se si deve fare come TeX che li ricompatta sempre.
`@InitialLanguage`	Il simbolo '**`@InitialLanguage`**' serve a adattare il comportamento di Lout in funzione del tipo di linguaggio. L'utilizzo di un linguaggio implica, per esempio, la scelta del modo in cui possono essere separate le sillabe e i nomi di alcune definizioni standard del documento.

Struttura del documento ordinario

Il contenuto di un documento scritto con Lout è racchiuso all'interno di uno o più ambienti specifici per il tipo di stile prescelto. Per esempio, nel caso del documento ordinario, si usano i comandi '**`@Text @Begin`**' e '**`@End @Text`**':

```
@SysInclude { doc }
@Document
    @InitialFont { Times Base 12p }
    …
//
@Text @Begin
    …
    …
@End @Text
```

Volendo, i due simboli possono essere posti anche su righe differenti, in modo da rendere più chiaro il loro significato, anche se questo è però contrario alla filosofia di Lout.

```
    …
@Text
@Begin
    …
    …
@End
@Text
```

Il simbolo '**`@Text`**' iniziale ha come argomento il testo del documento; in teoria questo potrebbe essergli fornito attraverso le parentesi graffe:

```
    …
@Text {
    …
    …
}
```

In pratica, questo modo di scrivere il sorgente Lout potrebbe essere troppo complicato; così, di fronte a oggetti di dimensioni molto grandi si preferisce utilizzare i delimitatori '**`@Begin`**' e '**`@End`**', nel modo mostrato.

In generale, un comando che può ricevere un oggetto delimitato dai simboli '**`@Begin`**' e '**`@End`**' può riceverlo anche se questo è racchiuso solo da parentesi graffe, mentre il contrario non è sempre possibile. In generale, si trova questa possibilità solo nei comandi che delimitano una struttura a larga scala.

L'ambiente '**`@Text`**' di questo tipo di documento può contenere anche delle sezioni e un'appendice, in modo simile a quello che viene mostrato nelle sezioni seguenti che fanno riferimento agli altri tipi di stile utilizzabile. In parte questo è già stato visto nello stesso esempio introduttivo.

Struttura della relazione tecnica

La relazione tecnica, ovvero lo stile '**`report`**', prevede dopo il preambolo l'inserimento facoltativo dell'ambiente '**`@Abstract`**'; successivamente prevede la presenza di uno o più ambienti '**`@Section`**', infine è ammessa la presenza di uno o più ambienti '**`@Appendix`**'.

```
@SysInclude { report }
@Report
    @Title {}
    …
//
@Abstract
    @Title {}
    …
@Begin
    …
    …
@End @Abstract
@Section
    @Title {}
    …
@Begin
    …
    …
@End @Section
    …
@Appendix
    @Title {}
    …
@Begin
    …
    …
@End @Appendix
    …
```

Si può intuire il senso di questi ambienti e il ruolo dell'opzione '**`@Title`**' che appare in ognuno di questi: la relazione tecnica può avere un riassunto introduttivo, si suddivide in sezioni e può terminare con un'appendice. A loro volta, le sezioni e le appendici si possono scomporre, nel modo che viene mostrato in seguito.

Struttura del libro

Il libro, ovvero lo stile '**`book`**', prevede dopo il preambolo l'inserimento facoltativo degli ambienti '**`@Preface`**' e '**`@Introduction`**'. Successivamente il documento viene suddiviso in capitoli, attraverso gli ambienti '**`@Chapter`**', e può concludersi con una serie di appendici.

```
@SysInclude { book }
@Book
    @Title {}
    …
//
@Preface
    @Title { Prefazione }
    …
@Begin
    …
    …
@End @Preface
@Introduction
    @Title { Introduzione }
    …
@Begin
    …
    …
@End @Introduction
@Chapter
    @Title {}
    …
@Begin
    …
    …
@End @Chapter
    …
@Appendix
    @Title {}
    …
@Begin
    …
    …
@End @Appendix
    …
```

Quello che si vede sopra è la struttura che potrebbe avere un documento di tipo '**book**' che include sia l'ambiente '**@Preface**' che '**@Introduction**'.

I capitoli possono suddividersi ulteriormente in sezioni, nello stesso modo in cui si possono inserire le sezioni nell'ambiente '**@Text**' quando si usa lo stile '**doc**'.

I capitoli potrebbero essere raggruppati in parti, ma non esistendo un ambiente del genere, si annota l'inizio di una nuova parte tra le opzioni del capitolo che si intende debba seguirla immediatamente. L'esempio seguente mostra il capitolo intitolato '**Primo approccio**' che si trova a essere il primo della parte '**Principianti**', indicata come '**Parte IV**'.

```
@Chapter
    @PartNumber { Parte IV }
    @PartTitle { Principianti }
    @Title { Primo approccio }
@Begin
    …
    …
@End @Chapter
```

Perché la suddivisione in parti venga presa in considerazione, è necessario che l'opzione '**@PartTitle**' abbia un argomento non vuoto, cioè disponga di un titolo. Se inoltre si vuole inserire del testo tra il titolo della parte e l'inizio del capitolo, occorre utilizzare l'opzione '**@PartText**' che prende come argomento il testo in questione.

```
@Chapter
    @PartNumber { Parte IV }
    @PartTitle { Principianti }
    @PartText {
        …
        …
        …
}
    @Title { Primo approccio }
@Begin
    …
    …
@End @Chapter
```

Struttura dei lucidi per lavagna luminosa

Le diapositive, ovvero lo stile '**slides**', prevede dopo il preambolo la suddivisione del documento in ambienti '**@Overhead**', che poi non possono contenere altre strutture a larga scala (in pratica non possono contenere sezioni o simili).

```
@SysInclude { slides }
@OverheadTransparencies
    @Title {}
    …
//
@Overhead
    @Title {}
    …
@Begin
    …
    …
@End Overhead
    …
```

Sottostrutture

L'ambiente '**@Text**' di un documento ordinario e i capitoli di un libro possono contenere delle sezioni, delimitate dai simboli '**@BeginSections**' e '**@EndSections**'. Si osservino gli esempi seguenti, dove nel primo caso vengono inserire delle sezioni all'interno di un documento ordinario, mentre nel secondo all'interno di un capitolo di un libro.

```
@SysInclude { doc }
@Document
    …
//
@Text @Begin
    …
@BeginSections
@Section
    @Title {}
    …
@Begin
    …
    …
@End @Section
    …
@EndSections
    …
@End @Text
```

```
@SysInclude { book }
@Book
    @Title {}
    …
//
    …
@Chapter
    @Title {}
    …
@Begin
    …
@BeginSections
@Section
    @Title {}
    …
@Begin
    …
    …
@End @Section
    …
@EndSections
    …
@End @Chapter
    …
```

All'interno delle sezioni, comprese quelle delle relazioni tecniche, è ammissibile la suddivisione in sottosezioni delimitate dai simboli '**@BeginSubSections**' e '**@EndSubSections**'.

```
@Section
    @Title {}
    …
@Begin
    …
@BeginSubSections
@SubSection
    @Title {}
    …
@Begin
    …
    …
@End @SubSection
    …
@EndSubSections
    …
@End @Section
```

Nello stesso modo funzionano anche le sotto-sottosezioni, attraverso la delimitazione dei simboli '**@BeginSubSubSections**' e '**@EndSubSubSections**'.

```
@SubSection
    @Title {}
    …
@Begin
    …
@BeginSubSubSections
@SubSubSection
    @Title {}
    …
@Begin
    …
    …
@End @SubSubSection
    …
@EndSubSubSections
    …
@End @SubSection
```

Lout non prevede ulteriori suddivisioni; comunque, anche le appendici possono essere suddivise in modo simile in sottoappendici e sotto-sottoappendici.

```
@Appendix
    @Title {}
    …
@Begin
    …
@BeginSubAppendices
@SubAppendix
    @Title {}
    …
@Begin
    …
@BeginSubSubAppendices
@SubSubAppendix
    @Title {}
    …
@Begin
    …
    …
@End @SubSubAppendix
    …
@EndSubSubAppendices
    …
```

```
@End @SubAppendix
    …
@EndSubAppendices
    …
@End @Appendix
```

Suddivisione del testo

Come già accennato in precedenza, Lout impone l'indicazione esplicita dell'inizio di un blocco di testo, ovvero un paragrafo. Gli spazi verticali non servono allo scopo come accade con LaTeX. In generale, si utilizzano i comandi '**@PP**' e '**@LP**'; il primo inizia un paragrafo normale, il secondo un paragrafo allineato a sinistra. La differenza sta nel fatto che normalmente '**@PP**' fa rientrare leggermente la prima riga, mentre il secondo no.

```
@PP
Lo stile «doc» permette una suddivisione del
testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni
```

L'esempio che si vede sopra è esattamente uguale, come risultato, a quello seguente:

```
@PP Lo stile «doc» permette una suddivisione del
testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni
```

La separazione del testo in paragrafi comporta normalmente l'inserzione di uno spazio verticale aggiuntivo tra la fine di uno e l'inizio del successivo. Per ottenere semplicemente l'interruzione di una riga (il ritorno a capo) si può utilizzare il comando '**@LLP**'.

```
@PP
Lo stile «doc» permette una suddivisione del
testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni;
@LLP
le sotto-sotto-sottosezioni non esistono.
```

L'esempio che si vede sopra è esattamente uguale, come risultato, a quello seguente:

```
@PP Lo stile «doc» permette una suddivisione del
testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni; @LLP le sotto-sotto-sottosezioni
non esistono.
```

Un paragrafo può essere messo in risalto (*display*) staccandolo dal resto del documento. Il comando '**@DP**' serve a ottenere un paragrafo senza il rientro della prima riga, un po' più staccato verticalmente da quello precedente.

```
@DP
Questo paragrafo risulta staccato meglio da quello
precedente.
```

Per aumentare questo distacco dal resto del testo, si possono usare più simboli '**@DP**' ripetutamente.

```
@DP
@DP
@DP
Questo paragrafo risulta molto staccato da quello
precedente.
```

Per richiedere espressamente il salto pagina in un punto del documento, si può usare il comando '**@NP**' il cui scopo è proprio quello di iniziare un paragrafo nuovo a partire dalla prossima colonna. Il paragrafo in questione non ha il rientro iniziale della prima riga.

```
@NP
Questo paragrafo inizia in una colonna, o in una pagina nuova.
```

Infine, il comando '**@CNP**' inizia un paragrafo che potrebbe essere spostato all'inizio della prossima colonna, o della prossima pagina, se non c'è abbastanza spazio per scrivere alcune righe.

È importante osservare che i simboli di questi comandi non prevedono argomenti e il testo del paragrafo che viene collocato dopo di questi non ne è legato in alcun modo. In pratica, il compito di '**@PP**' è quello di inserire uno spazio verticale aggiuntivo e memorizzare da qualche parte che il testo deve iniziare con una riga rientrata. Se si utilizza per due volte '**@PP**', si ottiene uno spazio di separazione verticale doppio.

Argomenti dei comandi e unità di misura

Fino a questo punto sono stati mostrati molti comandi di Lout senza descriverne il significato. Alcuni di questi richiedono un argomento composto dall'unione di più informazioni, come nell'esempio seguente,

```
@InitialFont { Times Base 20p }
@InitialBreak { ragged 1.2fx nohyphen }
```

dove ognuno dei due simboli mostrati richiede l'indicazione di tre argomenti raggruppati attraverso l'uso delle parentesi graffe.

Gli argomenti di un simbolo di Lout possono essere richiesti prima o dopo il simbolo stesso. Quando si tratta di informazioni numeriche che rappresentano una dimensione, queste sono intese essere espresse secondo un'unità di misura predefinita, oppure secondo l'unità stabilita da una lettera indicata subito dopo il numero. Per esempio, '**20p**' rappresenta 20 punti tipografici.

Tabella u93.36. Unità di misura principali di Lout.

Lettera	Unità di misura corrispondente
'**c**'	Centimetri.
'**i**'	Pollici ('**1i**' = '**2.54c**').
'**p**'	Punti tipografici ('**72p**' = '**1i**').
'**m**'	Quadratoni ('**12m**' = '**1i**').
'**f**'	La dimensione attuale del corpo.
'**s**'	La dimensione attuale di uno spazio (spaziatura).
'**v**'	La distanza attuale tra le righe.

Naturalmente, i valori che esprimono quantità non intere possono essere espressi utilizzando il punto di separazione tra la parte intera e quella decimale.

A volte, alcuni argomenti numerici devono essere conclusi con una lettera '**x**' (dopo l'indicazione dell'unità di misura). Intuitivamente si può associare questo fatto all'idea che si tratti di un valore che debba essere moltiplicato a qualcosa per ottenere il risultato, ovvero che si tratti di un dato relativo. Per esempio, il valore '**1.2fx**' del comando seguente rappresenta il 120 % dell'attuale dimensione dei caratteri (il corpo del carattere moltiplicato per 1,2):

```
@InitialBreak { ragged 1.2fx nohyphen }
```

La ragione precisa non è questa, ma la spiegazione approssimativa data può almeno essere utile per accettare la cosa temporaneamente, finché non si intende affrontare lo studio approfondito di Lout.

Rappresentazione simbolica della codifica

Come in tutti i sistemi di composizione tipografica, anche Lout ha un modo per rappresentare simbolicamente alcuni caratteri particolari. In precedenza si è accennato alla possibilità di inserire nel testo i caratteri speciali che Lout tende a interpretare in modo particolare, attraverso le stringhe letterali. Lout permette anche di usare dei simboli nella forma seguente:

```
@Char nome
```

Ciò permette di rappresentare qualunque carattere: sia l'alfabeto normale, sia i simboli di punteggiatura, sia qualunque altro simbolo speciale. Per esempio, '**@Char A**' è la lettera '**A**' maiuscola, mentre '**@Char a**' è la lettera '**a**' minuscola. La tabella u93.38 elenca alcuni dei comandi che possono essere utili per rappresentare le lettere accentate e altri caratteri importanti.

Tabella u93.38. Alcuni comandi per le lettere accentate di Lout.

`@Char aacute`	á	`@Char Aacute`	Á
`@Char acircumflex`	â	`@Char Acircumflex`	Â
`@Char agrave`	à	`@Char Agrave`	À
`@Char aring`	å	`@Char Aring`	Å
`@Char atilde`	ã	`@Char Atilde`	Ã

@Char adieresis	ä	@Char Adieresis	Ä
@Char ae	æ	@Char AE	Æ
@Char ccedilla	ç	@Char Ccedilla	Ç
@Char eacute	é	@Char Eacute	É
@Char ecircumflex	ê	@Char Ecircumflex	Ê
@Char egrave	è	@Char Egrave	È
@Char edieresis	ë	@Char Edieresis	Ë
@Char iacute	í	@Char Iacute	Í
@Char icircumflex	î	@Char Icircumflex	Î
@Char igrave	ì	@Char Igrave	Ì
@Char idieresis	ï	@Char Idieresis	Ï
@Char ntilde	ñ	@Char Ntilde	Ñ
@Char oacute	ó	@Char Oacute	Ó
@Char ocircumflex	ô	@Char Ocircumflex	Ô
@Char ograve	ò	@Char Ograve	Ò
@Char oslash	ø	@Char Oslash	Ø
@Char otilde	õ	@Char Otilde	Õ
@Char odieresis	ö	@Char Odieresis	Ö
@Char germandbls	ß		
@Char uacute	ú	@Char Uacute	Ú
@Char ucircumflex	û	@Char Ucircumflex	Û
@Char ugrave	ù	@Char Ugrave	Ù
@Char udieresis	ü	@Char Udieresis	Ü
@Char yacute	ý	@Char Yacute	Ý
@Char ydieresis	ÿ		

Quando si vuole rappresentare in questo modo una lettera accentata o un altro carattere tipografico speciale, come parte di una parola, si è costretti a inserire il comando relativo all'interno di parentesi graffe. Per comprendere il problema, si pensi alla possibilità di scrivere la parola «così» indicando la lettera '**i**' accentata con il comando '**@Char igrave**'. L'esempio seguente è errato:

```
cos@Char igrave                # errato
```

Infatti, Lout non è in grado di riconoscere il simbolo '**@Char**', dal momento che questo risulta attaccato ad altre lettere (e bisogna ricordare che per Lout il carattere '**@**' è una lettera come le altre). Il modo giusto di scrivere quella parola è quindi:

```
cos{ @Char igrave }
```

Oltre alla codifica normale, Lout mette a disposizione anche un alfabeto simbolico attraverso l'uso di comandi '**@Sym**'.

```
@Sym nome
```

Per conoscere i nomi che si possono utilizzare per ottenere le lettere greche e altri caratteri simbolici, deve essere letta la documentazione originale.

Caratteri da stampa

La scelta del carattere da stampa avviene prevalentemente attraverso una serie di comandi che riguardano la forma del carattere riferita allo stile attuale o l'indicazione precisa dello stile (ovvero della famiglia) e della forma.

```
@B { testo_in_neretto }
```

```
@I { testo_in_corsivo }
```

```
@BI { testo_in_neretto_corsivo }
```

```
@R { testo_in_tondo }
```

```
@S { testo_in_maiuscoletto }
```

Quelli che si vedono sono i comandi per ottenere una variazione della forma all'intero dello stile attuale del testo circostante. A questi comandi si affianca anche il comando per ottenere uno stile dattilografico, che pur non essendo semplicemente una variazione di forma, data la sua importanza nei documenti a carattere tecnico lo si abbina idealmente a questi per semplicità:

```
@F { testo_in_dattilografico }
```

Per cambiare in modo esplicito lo stile del carattere si può usare il comando '**@Font**' che richiede l'indicazione del nome dello stile, della forma e ovviamente del testo su cui intervenire:

```
{ stile forma } @Font { testo }
```

Gli stili più comuni sono: '**Times**', '**Helvetica**' e '**Courier**'. A questi si aggiungono anche delle specie simboliche, come '**Symbol**', solo che a questa si accede generalmente attraverso il comando '**@Sym**'.

La forma viene specificata attraverso una parola chiave che può essere: '**Base**', per indicare un carattere tondo chiaro; '**Slope**', per indicare una forma corsiva o inclinata (a seconda della disponibilità di quel tipo di stile); '**Bold**', per indicare il neretto; '**BoldSlope**', per indicare un neretto-corsivo. Naturalmente, la forma richiesta è ottenuta solo se lo stile scelto lo permette.

```
Lo stile { Courier BoldSlope } @Font { doc } permette una
suddivisione del testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni.
```

L'esempio che si vede sopra, serve a fare in modo che la parola '**doc**' sia resa con lo stile '**Courier**' in neretto-inclinato. Un risultato **simile** può essere ottenuto attraverso il comando '**@F**', nel modo seguente:

```
Lo stile @F { @BI { doc } } permette una
suddivisione del testo in sezioni, sottosezioni ed eventuali
sotto-sottosezioni.
```

I comandi di Lout per la definizione della forma non sono cumulativi, ed è per questo che esiste il comando '**@BI**'. L'esempio con cui si rende il carattere a larghezza fissa attraverso '**@F**' sembra contraddire questo, ma in realtà funziona perché si tratta di un comando riferito allo stile a cui poi si aggiunge un cambiamento di forma.

Corpo

Il corpo del carattere può essere modificato all'interno del documento (con il comando '**@Font**' utilizzato in modo differente da quanto visto finora), oppure può essere dichiarato nel preambolo che descrive gli aspetti generali dello stile prescelto. In ogni caso si rap-

presenta attraverso un numero seguito dall'unità di misura. Di solito si fa riferimento a punti tipografici, '**p**', dove per esempio '**12p**' rappresenta 12 punti tipografici (un punto = 1/72 di pollice).

Se il contesto lo consente, si possono indicare degli incrementi o delle riduzioni del valore precedente, dove per esempio '**+2p**' rappresenta l'incremento di due punti rispetto al carattere precedente e '**-1p**' rappresenta la riduzione di un punto. Nello stesso modo si possono indicare dei valori relativi, dove per esempio '**1.5f**' rappresenta una dimensione pari al 150 % del corpo utilizzato precedentemente.

Nel preambolo del documento si utilizza il comando '**@InitialFont**' che si trova in quasi tutti gli stili:

```
@InitialFont { stile forma corpo }
```

Nell'esempio introduttivo è stato utilizzato un carattere Times tondo chiaro da 24 punti:

```
@SysInclude { doc }
@Document
    @InitialFont { Times Base 24p }
    …
//
@Text @Begin
```

Quando si vuole modificare il corpo del carattere all'interno del documento, si usa il comando '**@Font**', con una delle due forme seguenti:

```
corpo @Font { testo }
```

```
{ stile forma corpo } @Font { testo }
```

L'esempio seguente mostra in che modo agire per ridurre leggermente (di due punti) il corpo di una parola:

```
L'operatore -2p @Font { AND } restituisce il valore booleano…
```

La stessa cosa avrebbe potuto essere ottenuta delimitando l'indicazione del corpo attraverso le parentesi graffe.

```
L'operatore { -2p } @Font { AND } restituisce il valore booleano…
```

L'esempio seguente mostra invece come è possibile modificare lo stile, la forma e il corpo simultaneamente.

```
La parola chiave { Courier Bold -1p } @Font { AND } serve a…
```

In precedenza è stato mostrato l'uso del comando '**@F**' per ottenere un carattere dattilografico; per la precisione si ottiene un carattere Courier chiaro con una dimensione pari a un punto in meno rispetto al corpo circostante. In pratica, '**@F**' è equivalente al comando:

```
{ Courier Base -1p } @Font { testo }
```

Display

Lout mette una cura particolare nella definizione di varie forme per fare risaltare un blocco di testo (*display*). Viene descritto brevemente l'elenco di quelle più comuni.

- Testo staccato.

```
@Display { testo }
```

```
@LeftDisplay { testo }
```

```
@LD { testo }
```

Il testo fornito come argomento del comando viene staccato in modo evidente dal paragrafo precedente e da quello successivo. La prima riga inizia senza rientri.

- Testo staccato e rientrato a sinistra.


```
@IndentedDisplay { testo }
```

```
@ID { testo }
```

Il testo fornito come argomento del comando viene staccato in modo evidente dal paragrafo precedente e da quello successivo, inoltre viene aumentato il margine sinistro.

- Testo staccato e rientrato a sinistra e a destra.

```
@QuotedDisplay { testo }
```

```
@QD { testo }
```

Il testo fornito come argomento del comando viene staccato in modo evidente dal paragrafo precedente e da quello successivo, inoltre viene aumentato il margine sinistro e anche quello destro.

Caratteristiche interne dei paragrafi

In precedenza, in occasione della descrizione della struttura di un documento Lout, è stato descritto l'uso dei comandi di separazione dei paragrafi ('**@LP**', '**@PP**' e altri). Questi non servono a definire le caratteristiche interne ai paragrafi, che invece possono essere specificate attraverso alcuni comandi da collocare nel preambolo, oppure attraverso '**@Break**'.

Il comando '**@Break**' può essere utilizzato per intervenire in un paragrafo il cui contenuto gli viene fornito come argomento, ma a sua volta non può apparire da solo. In pratica, negli esempi mostrati, '**@Break**' viene posto come argomento di un ambiente *display* di qualche tipo.

Interruzione e allineamento

Generalmente la suddivisione dei paragrafi in righe avviene in modo automatico, senza rispettare l'andamento del file sorgente. È possibile impedire la separazione in una certa posizione utilizzando il simbolo '**~**' in qualità di spazio orizzontale non interrompibile.

```
Il comando tar~cf~prova.tgz~/opt genera il file…
```

L'esempio mostra il modo in cui si può evitare che la descrizione di un comando del sistema operativo venga spezzato in corrispondenza degli spazi tra un argomento e l'altro.

Sia nel caso in cui la separazione in righe dei paragrafi venga ridefinita da Lout, sia quando si vogliono mantenere le interruzioni usate nel sorgente, si pone il problema di allineare il testo: a sinistra, in centro, a destra o simultaneamente a sinistra e a destra. Tutte queste cose si indicano attraverso una parola chiave che viene riconosciuta sia nel comando '**@Break**', sia nel comando '**@InitialBreak**' (il secondo si utilizza come opzione nel preambolo del documento). Tra queste si distinguono due gruppi importanti: quelle che terminano per '**ragged**', che si riferiscono a righe ricomposte da Lout, e quelle che terminano per '**lines**', che si riferiscono a righe interrotte esattamente come nel sorgente.

- '**ragged**' -- allineamento normale a sinistra;
- '**cragged**' -- allineamento centrato;
- '**rragged**' -- allineamento a destra;
- '**adjust**' -- allineamento simultaneo a sinistra e a destra;
- '**outdent**' -- allineamento simultaneo a sinistra e a destra con la prima riga sporgente dal lato sinistro;
- '**lines**' -- allineamento normale a sinistra rispettando le interruzioni di riga;
- '**clines**' -- allineamento centrato rispettando le interruzioni di riga;

- '**rlines**' -- allineamento a destra rispettando le interruzioni di riga.

Distanza tra le righe

La distanza tra le righe misura lo spazio che c'è tra la base di una riga e la base della successiva. Generalmente viene definito attraverso un valore relativo alla dimensione del carattere (al corpo), ma può essere indicato anche in modo assoluto. Per qualche motivo, il valore in questione deve essere terminato con il carattere '**x**'. Per esempio: '**1.20fx**' rappresenta una distanza di 1,2 volte il corpo del carattere (il 120 %); '**1.5vx**' rappresenta 1,5 volte la distanza preesistente, ma una notazione del genere può applicarsi solo quando esiste qualcosa di precedente a cui fare riferimento; '**14px**' rappresenta una distanza di 14 punti; '**1cx**' rappresenta una distanza di 1 cm.

Separazione in sillabe

La separazione in sillabe è un procedimento che dipende dal linguaggio, cosa che normalmente si seleziona nel preambolo del documento attraverso il comando '**@InitialLanguage**'. L'attivazione o meno della sillabazione dipende dal comando '**@Break**' o da '**@InitialBreak**' nel preambolo, attraverso una parola chiave: '**hyphen**' per attivarla e '**nohyphen**' per disattivarla.

Quando la sillabazione è attivata, si può utilizzare il simbolo '**&-**' per indicare a Lout la posizione di una possibile separazione delle parole. Per esempio, '**hard&-ware**' fa sì che se necessario la parola possa essere separata esattamente alla metà.

Raccogliere tutto assieme

Generalmente conviene regolare le caratteristiche dei paragrafi già nel preambolo, attraverso il comando '**@InitialBreak**':

```
@InitialBreak { allineamento distanza hyphen|nohyphen }
```

L'esempio seguente ripropone quanto già visto in precedenza riguardo alla definizione di un documento in forma di libro. Si può osservare la scelta di indicare la distanza tra le righe come un valore relativo riferito al corpo del carattere utilizzato.

```
@SysInclude { book }
@Book
    …
    @InitialBreak { adjust 1.2fx hyphen }
    …
//
```

All'interno del documento si può utilizzare il comando '**@Break**' nelle situazioni in cui ciò è possibile, per esempio in un ambiente che crea un blocco messo in risalto.

```
{ allineamento distanza hyphen|nohyphen } @Break { testo }
```

```
allineamento @Break { testo }
```

```
distanza @Break { testo }
```

```
hyphen|nohyphen @Break { testo }
```

L'esempio seguente mostra il caso di un paragrafo messo in risalto, nel quale viene ridotta la distanza tra le righe e si annulla la separazione in sillabe:

```
@QuotedDisplay { 0.8vx nohyphen } @Break {
Questa è un'informazione così importante
che facciamo in modo di rendervi difficile
la lettura }
```


Testo letterale

Il testo letterale può essere indicato utilizzando l'ambiente del comando '**@Verbatim**'. Possono essere usate due modalità equivalenti, che però hanno risvolti diversi nel contenuto che può avere l'ambiente in questione.

```
@Verbatim { testo_letterale }
```

```
@Verbatim @Begin
testo_letterale
    …
@End @Verbatim
```

Il primo dei due modi è adatto per le inclusioni brevi di testo, dove non si pongono problemi nell'uso delle parentesi graffe (queste possono essere contenute nel testo letterale, ma devono essere bilanciate correttamente); il secondo è l'alternativa per i blocchi di testo più lunghi e per quelle situazioni in cui le parentesi graffe possono creare dei problemi.

Se si utilizza il secondo modo di inclusione di testo letterale, il testo in questione non può contenere la parola '**@End**'.

Spesso, il testo incluso in modo letterale viene reso con un carattere dattilografico, come nell'esempio seguente o in quello successivo.

```
{ Courier Base } @Font @Verbatim @Begin
$ ls -l <invio>
…
@End @Verbatim
```

```
@F @Verbatim @Begin
$ ls -l <invio>
…
@End @Verbatim
```

Elenchi

Gli elenchi di Lout sono molto sofisticati, permettendo di gestire sia degli elenchi semplici, sia gli elenchi descrittivi, sia una lunga serie di elenchi puntati o numerati in vario modo. Tutti gli elenchi di Lout hanno in comune il simbolo che serve a concludere l'ambiente dell'elenco: '**@EndList**'.

Gli elenchi semplici sono solo un modo per staccare il testo evidenziandone così gli elementi. Questo si ottiene con l'ambiente introdotto dal simbolo '**@List**':

```
@List
@ListItem elemento
@ListItem elemento
    …
@EndList
```

Il risultato che si ottiene è costituito da una serie di paragrafi, uno per ogni punto, rientrati a sinistra e staccati verticalmente più dei paragrafi normali. L'esempio seguente dovrebbe rendere meglio l'idea.

```
@List
@ListItem { Tizio Tizi }
@ListItem {
Caio Cai, nato a Sferopoli il giorno … e trasferitosi
in altra città in seguito a… }
@ListItem { @B Sempronio Semproni }
@EndList
```

Gli elenchi puntati si ottengono con gli ambienti introdotti da uno tra i simboli '**@BulletList**', '**@StarList**' e '**@DashList**'. Si tratta rispettivamente di elenchi le cui voci sono precedute da un punto, un asterisco o un trattino.

```
@BulletList | @StarList | @DashList
@ListItem elemento
@ListItem elemento
    …
@EndList
```

Il risultato che si ottiene è lo stesso dell'elenco semplice, con l'aggiunta del puntino (o dell'asterisco o del trattino) nella parte sinistra all'inizio delle voci. L'esempio seguente è una variante di quello già presentato per l'elenco semplice, dove l'inizio delle voci è asteriscato.

```
@StarList
@ListItem { Tizio Tizi }
@ListItem {
Caio Cai, nato a Sferopoli il giorno … e trasferitosi
in altra città in seguito a… }
@ListItem { @B Sempronio Semproni }
@EndList
```

Gli elenchi numerati si ottengono con gli ambienti introdotti da uno tra i simboli '**@NumberedList**', '**@RomanList**', '**@UCRomanList**', '**@AlphaList**', '**@UCAlphaList**', e dalla serie parallela '**@ParenNumberedList**', '**@ParenRomanList**', '**@ParenUCRomanList**', '**@ParenAlphaList**', '**@ParenUCAlphaList**'. I due raggruppamenti di simboli Lout si riferiscono a numerazioni normali o numerazioni tra parentesi ('**Paren**'); i simboli il cui nome contiene la parola '**Roman**' rappresentano una numerazione romana; i simboli il cui nome contiene la parola '**Alpha**' rappresentano una numerazione alfabetica; il prefisso '**UC**' specifica che si tratta di una numerazione (romana o alfabetica) maiuscola.

```
@NumberedList | @RomanList | @UCRomanList | @AlphaList |
@UCAlphaList
@ListItem elemento
@ListItem elemento
    …
@EndList
```

```
@ParenNumberedList | @ParenRomanList | @ParenUCRomanList↩
↪ | @ParenAlphaList | @ParenUCAlphaList
@ListItem elemento
@ListItem elemento
    …
@EndList
```

Per la realizzazione di elenchi composti, dove un punto si articola in sottopunti, basta inserire un elenco all'interno di una voce, per esempio nel modo seguente:

```
@NumberList
@ListItem { Tizio Tizi }
@ListItem { Caio Cai
@BulletList
@ListItem { nato a Sferopoli il… }
@ListItem { residente a… }
@EndList }
@ListItem { @B Sempronio Semproni }
@EndList
```

Gli elenchi descrittivi permettono di specificare ciò che si vuole usare per indicare ogni voce. In pratica si tratta di una stringa che rappresenta un'etichetta, ovvero una sorta di titolo della voce. Lout mette a disposizione diversi simboli in funzione della distanza che si intende lasciare tra l'inizio dell'etichetta e il blocco di testo a cui questa fa riferimento: '**@TaggedList**', '**@WideTaggedList**' e '**@VeryWideTaggedList**'.

```
@TaggedList | @WideTaggedList | @VeryWideTaggedList
@TagItem { etichetta } { elemento }
@TagItem { etichetta } { elemento }
    …
@EndList
```


Quando per qualche motivo si ha a che fare con etichette troppo lunghe, o comunque può risultare inopportuno fare iniziare il blocco di testo sulla stessa riga dell'etichetta, al posto del simbolo '**@TagItem**' per introdurre le voci si può usare '**@DropListItem**'.

```
@TaggedList | @WideTaggedList | @VeryWideTaggedList
@DropListItem { etichetta } { elemento }
@DropListItem { etichetta } { elemento }
    …
@EndList
```

Note

Lout organizza in modo molto raffinato le note a piè pagina, le note finali e le note a margine. Qui vengono mostrate solo le caratteristiche essenziali.

Note a piè pagina e note finali

La distinzione tra note a piè pagina e note finali sta nel fatto che le prime appaiono nella stessa pagina in cui si trova il loro riferimento, o al massimo in quella successiva, mentre le seconde si collocano alla fine del documento (o alla fine del capitolo).

```
@FootNote [ @Location { ColFoot | PageFoot } ] { testo }
```

```
@EndNote { testo }
```

Come si può intuire, il comando '**@FootNote**' riguarda l'inserimento di una nota a piè pagina, mentre '**@EndNote**' di una nota alla fine del documento. In particolare, la nota a piè pagina può essere collocata alla fine della colonna, o alla fine della pagina; in questo ultimo caso può occupare tutte le colonne della pagina. Utilizzando l'opzione '**@Location**' con l'argomento '**ColFoot**' si ottiene una nota che si espande orizzontalmente solo nello spazio della colonna, mentre con '**PageFoot**', si vuole fare in modo che la nota si allarghi per tutto lo spazio orizzontale della pagina.

```
@FootNote
    @Location { PageFoot }
{ Questa è una nota a piè pagina che si espande
orizzontalmente occupando tutta la larghezza della
pagina, anche se questa è suddivisa in più di una
colonna. }
```

```
@EndNote { Questa è una nota alla fine del
documento. }
```

In generale, è sconsigliabile l'uso simultaneo di note a piè pagina e note a fine documento, in quanto non è possibile distinguere facilmente i riferimenti che vengono collocati nella composizione finale.

Note a margine

Le note a margine sono annotazioni molto brevi che si collocano sullo spazio del margine sinistro o del margine destro della pagina. Per ottenerle si utilizzano i comandi '**@LeftNote**' o '**@RightNote**' a seconda che si voglia la nota sul margine sinistro o sul margine destro. Se si inseriscono in un documento che distingue tra pagine destre e sinistre, si possono utilizzare i comandi '**@OuterNote**' e '**@InnerNote**' per indicare rispettivamente le note sul margine esterno o sul margine interno.

```
@LeftNote { nota_sul_margine_sinistro }
```

```
@RightNote { nota_sul_margine_destro }
```

```
@InnerNote { nota_sul_margine_interno }
```

```
@OuterNote { nota_sul_margine_esterno }
```

Figure e tabelle fluttuanti

Come per LaTeX, le figure e le tabelle possono essere parte del testo normale, oppure possono essere inserite in un involucro che le rende fluttuanti. L'involucro in questione è praticamente identico nei due casi, a parte il simbolo iniziale che serve a distinguere la numerazione delle figure da quella delle tabelle. Di conseguenza, l'oggetto che compone la figura o la tabella all'interno di questo involucro, può essere qualunque cosa, in base alle intenzioni dell'autore.

```
@Figure | @Table
    @Location { collocazione }
    @OnePage { Yes | No }
    @FullPage { Yes | No }
    @CaptionPos { Above | Below }
    @Caption { didascalia }
oggetto
```

Come accennato, il simbolo '**@Figure**' rappresenta un involucro fluttuante per una figura, mentre '**@Table**' è quello da usare per una tabella. Le opzioni che si vedono nello schema sintattico sono tutte facoltative:

- il simbolo '**@Location**' permette di definire la collocazione fluttuante della figura o della tabella, attraverso un argomento composto da una parola chiave:
  - '**PageTop**' -- l'oggetto deve essere collocato all'inizio della pagina successiva,
  - '**PageFoot**' -- l'oggetto deve essere collocato alla fine della pagina corrente,
  - '**ColTop**' -- l'oggetto deve essere collocato all'inizio della colonna successiva,
  - '**ColFoot**' -- l'oggetto deve essere collocato alla fine della colonna corrente,
  - '**ColEnd**' -- l'oggetto deve essere collocato in una colonna alla fine del documento (o del capitolo),
  - '**AfterLine**' -- l'oggetto deve essere collocato esattamente dove si trova (nella riga successiva in base al risultato della composizione),
  - '**TryAfterLine**' -- l'oggetto deve essere collocato esattamente dove si trova, a meno che lo spazio sia insufficiente, perché in tal caso viene spostato all'inizio della colonna successiva,
  - '**Display**' -- l'oggetto deve essere messo in risalto e collocato esattamente dove si trova,
  - '**Raw**' -- l'oggetto non deve essere fluttuante e deve rimanere com'è (serve a inserire delle immagini all'interno delle celle di una tabella);
- il simbolo '**@OnePage**' permette di definire se si vuole che l'oggetto fluttuante debba rimanere intero o se possa essere diviso tra una pagina e la successiva (o tra colonne), il valore predefinito varia in funzione del tipo di collocazione prescelto;
- il simbolo '**@FullPage**' permette di definire se si vuole che l'oggetto fluttuante debba occupare da solo lo spazio di una pagina, oppure se questo possa essere condiviso con il testo;
- il simbolo '**@Caption**' permette di indicare la didascalia dell'oggetto;
- il simbolo '**@CaptionPos**' permette di stabilire la posizione in cui deve apparire la didascalia (in alto, '**Above**', o in basso, '**Below**').

Figure

Come già spiegato, qualunque oggetto può essere una figura. Di solito questo oggetto è ottenuto con il comando '**@Fig**', che qui non viene descritto. In alternativa si mostra in che modo inserire del testo letterale, che alle volte può servire per lo scopo. Si osservi l'esempio seguente:

```
@Figure
    @Location { TryAfterLine }
    @OnePage { Yes }
    @FullPage { No }
    @CaptionPos { Below }
    @Caption { @I { standard input } e @I { standard output } }
@F @Verbatim @Begin
                    .-----------.
                    |           |-----------> STDOUT
STDIN   ----------->| Programma |
                    |           |-----------> STDERR
                    '-----------'
@End @Verbatim
```

La figura dell'esempio rappresenta uno schema costruito attraverso del testo letterale utilizzando un carattere dattilografico. Questa dovrebbe essere collocata immediatamente sotto il punto in cui appare nel sorgente; se non dovesse esserci spazio sufficiente fino alla fine della pagina, verrebbe spostata all'inizio di quella successiva.

Nel caso si realizzino figure nel modo proposto dall'esempio, occorre fare attenzione a non inserire delle tabulazioni nel sorgente, perché verrebbero interpretate in un modo diverso da quello che può fare il programma che si utilizza per la sua scrittura. Eventualmente si può filtrare il sorgente con il comando '**expand -8**', in modo da ottenere la trasformazione delle tabulazioni in spazi normali.

Tabelle

Le tabelle di Lout possono essere molto sofisticate. Anche in questo caso vale lo stesso discorso fatto per le figure, dove l'oggetto che si intende fornire per la descrizione della tabella può essere qualsiasi cosa, anche se in pratica si tratta quasi sempre di un comando '**@Tab**'. Qui si intende mostrare solo un uso elementare; dettagli maggiori possono essere trovati nella documentazione originale.

Per poter utilizzare le tabelle di Lout, cioè quelle che si ottengono con il comando '**@Tab**', occorre includere uno stile aggiuntivo prima della dichiarazione del tipo di documento fondamentale:

```
@SysInclude { tab }
```

Per esempio, nel caso del documento ordinario si dovrebbe iniziare nel modo seguente:

```
@SysInclude { tab }
@SysInclude { doc }
@Document
    …
//
```

Quello che si vede sotto è lo schema sintattico di una tabella estremamente semplificata. Si noti in particolare il fatto che le colonne sono distinte da una lettera alfabetica maiuscola.

```
@Tab
    @Fmta { @Col A ! @Col B ! … }
{
@Rowa A { cella_1.1 } B { cella_1.2 } …
@Rowa A { cella_2.1 } B { cella_2.2 } …
…
}
```

Utilizzando questo schema semplificato, si ottengono delle tabelle senza linee (né verticali, né orizzontali) per evidenziare o abbellire le sue parti. Di solito, per quanto semplice sia la tabella, si ha almeno l'esigenza di utilizzare delle linee orizzontali per evidenziare le righe che compongono l'intestazione delle colonne e per segnalare la fine della tabella. Per questo si devono usare delle parole chiave aggiuntive che si collocano tra gli argomenti di '**@Rowa**' cioè dei comandi che descrivono le righe.

```
@Rowa
    above { single }
    A { cella } B { cella } …
    below { single }
```

L'opzione '**above { single }**' inserisce una linea orizzontale sopra la riga a cui si riferisce, mentre '**below { single }**' la inserisce sotto. Supponendo di voler ottenere una tabella come quella schematizzata qui sotto,

Parametro LOC	Posizione corrispondente
h	posizione attuale
t	superiore
b	inferiore
p	pagina

Esempio di tabella.

il codice necessario per Lout potrebbe essere quello seguente:

```
@Table
    @Location { TryAfterLine }
    @OnePage { Yes }
    @FullPage { No }
    @CaptionPos { Below }
    @Caption { Esempio di tabella }
@Tab
    @Fmta { @Col A ! @Col B }
{
@Rowa
    above { single }
    A { Parametro LOC }
    B { Posizione corrispondente }
    below { single }
@Rowa
    A { h }
    B { posizione attuale }
@Rowa
    A { t }
    B { superiore }
@Rowa
    A { b }
    B { inferiore }
@Rowa
    A { p }
    B { pagina }
    below { single }
}
```

Indici e riferimenti incrociati

Lout crea automaticamente una serie di riferimenti incrociati. I più comuni sono quelli dei piè pagina e quelli degli indici. Lout richiede l'elaborazione ripetuta di un sorgente per sistemare proprio questi indicatori; in particolare, a differenza di LaTeX (che in generale richiede tre passaggi, o quattro se si inserisce BibTeX), non si può prevedere quante volte debba essere rifatta la composizione.

Riferimenti nel testo

Come accennato, le note a piè pagina e quelle alla fine del documento sono un esempio di inserzione nel testo di riferimenti a qualcosa che appare altrove. Quando si vuole indicare un riferimento a qualcosa di diverso, si usano i comandi '**@PageOf**' e '**@NumberOf**'.

```
@PageOf { nome_del_riferimento }
```

```
@NumberOf { nome_del_riferimento }
```

Il primo dei due viene rimpiazzato da Lout con il numero della pagina in cui si trova il riferimento indicato, mentre il secondo mostra il numero del capitolo o della sezione relativa. Per esempio, se da qualche parte è stato dichiarato il riferimento denominato '**presentazione**', vi si può fare riferimento come in questo estratto:

```
Come accennato in precedenza (a pagina @PageOf { presentazione }),
la matematica non è un'opinione.
```

Successivamente, il testo che si vede sopra si trasforma nella composizione in qualcosa di simile a quello che segue:

```
Come accennato in precedenza (a pagina 11),
la matematica non è un'opinione.
```


La dichiarazione di un riferimento (in altri termini di un'etichetta) può essere fatta con il comando '**@PageMark**', oppure con l'opzione '**@Tag**' che si può inserire all'inizio dei capitoli, delle sezioni, delle appendici e delle loro strutture inferiori.

```
@PageMark { nome_del_riferimento }
```

```
@Chapter|@Section|@SubSection|@SubSubSection
    @Title { titolo }
    @Tag { nome_del_riferimento }
    …
@Begin
    …
```

Riprendendo l'esempio precedente, si vede come potrebbe essere dichiarato un riferimento raggiungibile attraverso il comando '**@PageOf**':

```
Attenzione: @MarkOf { presentazione } la matematica
non è un'opinione perché…
```

Naturalmente, nel testo risultante dalla composizione non si vede la dichiarazione.

Lout è un po' rigido nell'uso di questi riferimenti: attraverso '**@PageOf**' si può fare riferimento alla pagina che contiene sia un riferimento dichiarato con '**@PageMark**', sia un riferimento dichiarato all'inizio della struttura per mezzo dell'opzione '**@Tag**', ma con '**@NumberOf**' si può solo fare riferimento solo a quanto dichiarato con l'opzione '**@Tag**'.

I nomi utilizzati per indicare i riferimenti devono essere univoci. Generalmente si utilizzano nomi composti solo da lettere alfabetiche ed eventualmente dal punto (come suggerisce l'autore di Lout). Se ce ne fosse la necessità, si può sempre delimitare questi nomi attraverso l'uso delle virgolette.

Indice generale e indice analitico

A seconda dello stile del documento prescelto, l'indice generale viene incluso o meno, in modo automatico. Questo comportamento può essere modificato ritoccando il file di stile, oppure, creando uno stile personalizzato. Gli stili standard prevedono al massimo la stampa dell'indice generale; se si desidera ottenere un indice delle figure o delle tabelle occorre intervenire nello stile in ogni caso.

Anche l'indice analitico viene aggiunto automaticamente se il tipo di documento è adatto per questo, però in tal caso dipende dall'autore l'inserimento dei riferimenti che lo generano. Lout consente l'uso di una grande varietà di tecniche per ottenere un indice analitico veramente buono. Qui viene mostrato l'essenziale.

```
chiave_per_ordinamento @Index { voce }
```

Quello che si vede è la sintassi minima per inserire un riferimento nel testo che poi si traduce in una voce nell'indice analitico. La voce in questione viene mostrata utilizzando quanto indicato alla destra del simbolo '**@Index**', ordinata in base alla chiave indicata alla sua sinistra.

Il principio è che l'ordine (alfabetico) con cui devono essere ordinate le voci potrebbe essere diverso da quello che si viene a generare utilizzando direttamente il contenuto delle voci. Per fare un esempio tra le tante situazioni che si possono creare, la voce '**Decimo**' potrebbe dover apparire prima di '**De Tizi**', ma utilizzandole così come sono, lo spazio tra '**De**' e '**Tizi**' farebbe sì che questa ultima voce appaia per prima. Per questo è necessario specificare una voce alternativa da utilizzare per l'ordinamento. Per convenzione, oltre che per evitare imprevisti, è bene limitarsi all'uso delle sole lettere alfabetiche minuscole, non accentate, ed è per questo che nella sintassi non sono state usate le parentesi graffe per racchiudere l'argomento a sinistra del simbolo '**@Index**'.

Volendo realizzare un indice analitico strutturato in voci e sotto-voci, si può utilizzare il comando '**@SubIndex**', con l'aggiunta del comando '**SubSubIndex**' per una suddivisione ulteriore.

```
chiave_per_ordinamento @SubIndex { voce }
```

```
chiave_per_ordinamento @SubSubIndex { voce }
```

Le sotto-voci sono interessanti in quanto riferite a una voce di livello precedente. La documentazione di Lout suggerisce di utilizzare delle chiavi strutturate, ottenute a partire dalla chiave della voce principale unendo un punto e aggiungendo un'estensione opportuna.

```
Tizio Tizi tiziotizi @Index { Tizio Tizi } è stato lo
scopritore di…
…
Tizio Tizi tiziotizi.origini @SubIndex { origini } era figlio di
Pinco Pallino e di…
```

L'esempio dovrebbe mostrare in maniera sufficientemente chiara il concetto: da qualche parte del testo si parla di '**Tizio Tizi**' e lì viene inserito un riferimento; da un'altra parte si parla sempre di lui, ma in particolare si descrivono le sue origini. In pratica, la voce '**origini**' dipende da '**Tizio Tizi**' e opportunamente la chiave di ordinamento fa in modo che questa risulti successiva.

Seguendo la logica dell'esempio mostrato, se si scrive un capitolo su '**Tizio Tizi**', potrebbe non avere significato un riferimento a una pagina in cui si parla di questa persona, mentre ci si troverebbe ad avere solo delle sottoclassificazioni (origini, vita, morte, ecc.). Volendo indicare una voce senza che con questa si ottenga il numero della pagina corrispondente, si può utilizzare il comando '**@RawIndex**'.

```
chiave_per_ordinamento @RawIndex { voce }
```

L'esempio già mostrato potrebbe essere modificato convenientemente nel modo seguente:

```
Tizio Tizi tiziotizi @RawIndex { Tizio Tizi } è stato lo
scopritore di…
…
Tizio Tizi tiziotizi.origini @SubIndex { origini } era figlio di
Pinco Pallino e di…
…
Tizio Tizi è nato tiziotizi.nascita @SubIndex { nascita }
in un paesino sperduto…
…ed è morto tiziotizi.morte @SubIndex { morte } il giorno
…
```

Problemi connessi alla generazione dei riferimenti incrociati

Quando si modifica un documento che fa uso di riferimenti incrociati di qualunque tipo (praticamente sempre), prima di riavviare l'eseguibile '**lout**' per ottenerne la composizione sarebbe opportuno eliminare i file transitori che vengono creati da questo. Supponendo di lavorare con il file '`pippo`', occorrerebbe eliminare il file '`pippo.ld`' e '`lout.li`'.

Diversamente, è probabile che una sola passata basti a ottenere il formato finale senza ottenere segnalazioni di errore, ma i riferimenti aggiunti nel documento potrebbero essere errati o mancare del tutto.

Localizzazione

I problemi di localizzazione di un documento riguardano generalmente le definizioni standard di alcune componenti tipiche (capitolo, appendice, indice, ecc.) e la sillabazione. Per attuare questo con Lout si utilizza l'opzione '**@InitialLanguage**' nel preambolo del documento, mentre a livelli inferiori si possono circoscrivere delle eccezioni.

```
@SysInclude { book }
@Book
    …
    @InitialLanguage { Italian }
    …
//
```


L'esempio mostra in che modo potrebbe essere definito il linguaggio «italiano» per tutto un documento (in questo caso un libro).

All'interno del testo è possibile alterare il linguaggio generale attraverso il comando '**@Language**':

```
linguaggio @Language { testo }
```

Per esempio, per indicare che una frase è scritta in tedesco si potrebbe fare come nell'esempio seguente:

```
La nonna disse: German @Language { Wer bekommt die Torte? }
```

Configurazione di una localizzazione

Nel momento in cui viene scritto questo capitolo, le versioni di Lout che si trovano comunemente in circolazione non dispongono del linguaggio italiano. Per prepararselo occorre intervenire su alcuni file: '`/usr/lib/lout/include/langdefs`', '`/usr/lib/lout/data/standard.ld`' e '`/usr/lib/lout/hyph/italian.lh`'. L'ultimo di questi serve per definire le regole della sillabazione e di solito viene creato a partire da quello di un altro paese. Questo file è diviso in due parti, dove la seconda, cioè quella che indica precisamente le regole della separazione in sillabe, può essere ricopiata dal file corrispondente utilizzato per LaTeX.

Personalizzazione dello stile

Invece di utilizzare uno degli stili standard di Lout, si può creare il proprio, di solito modificandone uno preesistente. Quando si crea uno stile riferito a un documento particolare, può darsi che il file relativo venga tenuto assieme a quello del documento stesso; in tal caso può convenire di utilizzare un comando di inclusione diverso dal solito. Supponendo di voler creare una variante dello stile '**book**', si potrebbe copiare il file corrispondente, '`/usr/lib/lout/include/book`', nella directory di lavoro del documento e chiamarlo '`libro`'. In questo modo, l'inizio del documento potrebbe essere organizzato nel modo seguente:

```
@Include { libro }
@Book
//
@Chapter @Begin
    …
```

Si osservi l'uso del comando '**@Include**' che si riferisce alla directory corrente o a un percorso assoluto (se indicato).

Nelle sezioni seguenti si accenna all'organizzazione di questo file di stile. Per modificarlo basta intervenire negli argomenti delle opzioni indicate; anche senza conoscere precisamente i dettagli, si dovrebbe riuscire nell'intento utilizzando semplicemente l'intuito.

Inclusione di altri stili

Nella prima parte del file di stile si incontra una serie di inclusioni possibili per l'aggiunta di altri stili.

```
###############################################################################
#                                                                             #
#  @SysInclude commands for standard packages.                                #
#                                                                             #
###############################################################################

  @SysInclude { fontdefs }                  # font definitions
  @SysInclude { langdefs }                  # language definitions
  @SysInclude { dl }                        # DocumentLayout package
  @SysInclude { bookf }                     # BookLayout extension
# @SysInclude { tab }                       # @Tab table formatter
# @SysInclude { eq }                        # @Eq equation formatter
# @SysInclude { fig }                       # @Fig advanced graphics
# @SysInclude { graph }                     # @Graph graph drawing
# @SysInclude { cprint }                    # @CPrint C and C++ programs
# @SysInclude { pas }                       # @Pas Pascal programs
```

Come si vede, le inclusioni che non sono necessarie appaiono commentate. Potrebbe essere conveniente togliere il commento da qualcosa, per esempio l'inclusione dello stile '**tab**' in modo da consentire la realizzazione di tabelle attraverso il comando '**@Tab**'.

Dopo le inclusioni standard appare l'inserimento predefinito dello stile '**mydefs**', nel caso fosse presente nella directory di lavo-

ro nel momento della composizione. In pratica, questo è il nome convenzionale di un file da usare per la personalizzazione aggiuntiva.

```
##############################################################################
#                                                                            #
#  @Include command for reading personal definitions from current directory. #
#                                                                            #
##############################################################################

  @Include { mydefs }
```

Veste grafica del documento

Nell'ultima parte del file di stile si definisce una serie di cose che riguardano la veste grafica del documento. Nei file di configurazione standard sono riportate tutte le opzioni disponibili con gli argomenti predefiniti, commentate attraverso il carattere '#' e descritte.

```
@Use { @DocumentLayout
  # @InitialFont        { Times Base 12p        } # initial font
  # @InitialBreak       { adjust 1.20fx hyphen  } # initial break
  # @InitialSpace       { lout                  } # initial space style
  # @InitialLanguage    { English               } # initial language
    …
}
```

Particolarità del tipo di documento

A seconda dello stile originale da cui si è partiti per realizzare il proprio, l'ultima parte potrebbe essere diversa. Per esempio, nel caso del libro, questa comincia così:

```
@Use { @BookLayout
  # @TitlePageFont              { Helvetica Base } # title page font (not size)
  # @SeparateIntroNumbering     { Yes            } # separate intro page numbers
  # @ChapterStartPages          { Any            } # Any, Odd, or Even
  # @ReferencesBeforeAppendices { No             } # pos of ref list
    …
}
```

Parte conclusiva

La parte finale del file della configurazione dello stile viene lasciato normalmente così come si trova.

```
##############################################################################
#                                                                            #
#  @Database (and @SysDatabase) clauses go here.                             #
#                                                                            #
##############################################################################

@SysDatabase @RefStyle { refstyle }              # reference printing styles
```

Riferimenti

- Jeffrey H. Kingstom, *A User's Guide to the Lout Document Formatting System*
- Jeffrey H. Kingstom, *A Practical Introduction to the Lout Document Formatting System*

[1] **Lout** GNU GPL

Introduzione a LyX

Creazione di un documento 705
Struttura e stile 706
Modelli di documento 707
Automatismi 707
Riferimenti 707

LyX [1] è un sistema di composizione tipografica visuale, che si avvale principalmente di LaTeX per generare il risultato finale. Sono disponibili almeno due versioni di LyX differenti, in base al tipo di librerie grafiche utilizzate: Qt oppure XForms (nel secondo caso si tratta di software proprietario).

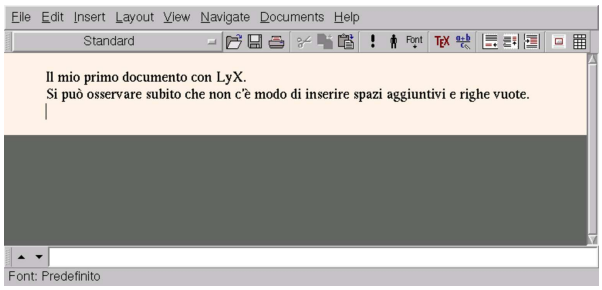
LyX viene definito dai suoi sviluppatori come un sistema di scrittura WYSIWYM, ovvero, *What you see is what you mean*, a differenza dei sistemi visuali comuni (definiti WYSIWYG, ovvero *What you see is what you get*), perché ciò che si vede sullo schermo dà solo l'idea del risultato finale.

In generale, si può considerare LyX come il tramite per coloro che sono spaventati dallo scrivere un documento senza l'aiuto di uno strumento visuale, anche se da un punto di vista operativo, alla fine, la scrittura diretta di un sorgente (LaTeX, Lout, SGML, XML o altro), è sempre la scelta migliore. In questo modo ci si può abituare all'idea e poi il passaggio è meno traumatico. Questa precisazione è bene farla, perché non ci si può aspettare da LyX la stabilità di funzionamento che si può avere scrivendo direttamente un sorgente per la composizione differita; pertanto diventa difficile trovare delle motivazioni migliori a quella espressa per usare LyX piuttosto di un altro sistema di scrittura visuale.

Creazione di un documento

Si avvia LyX con l'eseguibile '**lyx**', che può essere avviato senza argomenti. Per creare un documento nuovo, basta aprire il menù *File* e selezionare la voce *New*. La figura u94.1 mostra in che modo si può presentare LyX mentre si scrive un file per la prima volta. Mentre si scrive, si può osservare subito che non si possono inserire spazi aggiuntivi e nemmeno righe vuote.

Figura u94.1. Scrittura di un documento nuovo con LyX.



Si può salvare il documento selezionando la voce *Save as* dal menù *File*, specificando poi il nome del file, che ha preferibilmente l'estensione '`.lyx`'.

Il file che si ottiene è un file di testo che ha una vaga somiglianza con TeX. Quanto si vede nella figura u94.1 si traduce in pratica nel testo seguente:

```
#LyX 1.3 created this file. For more info see http://www.lyx.org/
\lyxformat 221
\textclass article
\language english
\inputencoding auto
\fontscheme default
\graphics default
\paperfontsize default
\papersize Default
\paperpackage a4
\use_geometry 0
\use_amsmath 0
\use_natbib 0
```

```
\use_numerical_citations 0
\paperorientation portrait
\secnumdepth 3
\tocdepth 3
\paragraph_separation indent
\defskip medskip
\quotes_language english
\quotes_times 2
\papercolumns 1
\papersides 1
\paperpagestyle default

\layout Standard

Il mio primo documento con LyX.
\layout Standard

Si può osservare subito che non c'è modo di inserire spazi aggiuntivi e
 righe vuote.
\layout Standard

\the_end
```

Il documento può essere stampato selezionando la voce *Print* dal menù *File*. Per arrivare alla stampa, LyX usa LaTeX in modo trasparente, con un file che potrebbe essere simile a quello seguente:

```
%% LyX 1.3 created this file.  For more info, see http://www.lyx.org/.
%% Do not edit unless you really know what you are doing.
\documentclass[english]{article}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}

\makeatletter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% LyX specific LaTeX commands.
\providecommand{\LyX}{L\kern-.1667em\lower.25em\hbox{Y}\kern-.125emX\@}

\usepackage{babel}
\makeatother
\begin{document}
Il mio primo documento con \LyX{}.

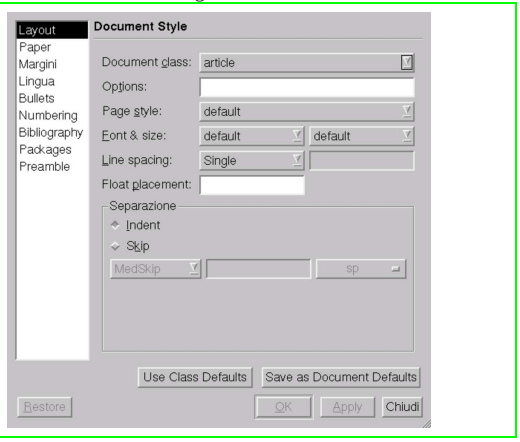
Si può osservare subito che non c'è modo di inserire spazi aggiuntivi
e righe vuote.
\end{document}
```

Di conseguenza, ciò che si ottiene con la stampa è esattamente il risultato della composizione di questo file con LaTeX.

Struttura e stile

Dal menù *Layout* è possibile accedere a funzionalità che cambiano l'aspetto del documento. La cosa più importante che si deve stabilire del documento che si va a scrivere è la sua struttura, secondo dei modelli prestabiliti, attraverso la voce *Document*. La figura u94.4 mostra la finestra di inserimento che si ottiene, dove si può osservare in alto la presenza di un menù a scomparsa con un elenco di stili generali.

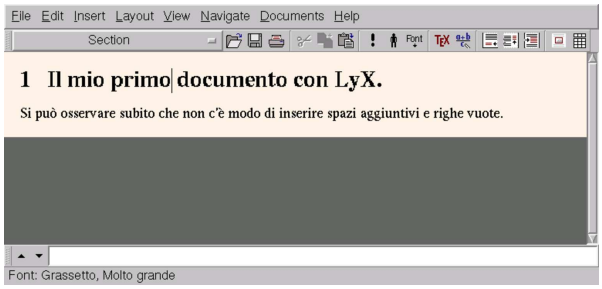
Figura u94.4. Caratteristiche generali del documento.



Selezionando uno stile complessivo differente, la struttura del documento cambia. Per esempio, il tipo di documento può ammettere una suddivisione in capitoli, oppure solo in sezioni di minore importanza, oppure può essere sprovvisto di suddivisioni. Ancora più importante di questo, la scelta dello stile complessivo dichiara anche il modo in cui si vuole arrivare alla composizione finale. Nell'esempio iniziale della figura u94.1 si fa riferimento allo stile generale predefinito, corrispondente allo stile '**`article`**' di LaTeX; in questo modo la composizione passa per la trasformazione in LaTeX. Se si osservano le voci che appaiono nel menù a scomparsa della voce *Layout* già descritta (figura u94.4), si possono notare alcuni stili riferiti espressamente a DocBook: il loro utilizzo implica una composizione che utilizza strumenti relativi a DocBook, pertanto viene generato un sorgente SGML e non più LaTeX.

Durante la scrittura del documento, il testo che viene inserito viene associato a uno stile predefinito, in relazione al tipo di documento scelto. Se si guarda nuovamente la figura u94.1, si può notare che sotto alle voci del menù normale appare un menù a scomparsa, sul quale si legge il nome '**`Standard`**'. Il significato è semplice: il testo sul quale si trova il cursore è associato allo stile standard. Basta mettere il cursore su un blocco di testo, per esempio la prima riga, scegliere una voce differente da questo menù a scomparsa per ottenerne l'adattamento al suo stile. Nella figura u94.5 è stato associato il testo della prima riga allo stile '**`Section`**', trasformandolo così nel titolo di una sezione.

Figura u94.5. Modifica dello stile associato a un blocco di testo.



Naturalmente, nell'ambito delle possibilità dello stile a cui è associato il blocco di testo, è possibile intervenire per modificare localmente una porzione di questo o anche il paragrafo nel suo complesso, ma per questo si deve agire sempre nel menù *Layout*, selezionando altre voci.

Modelli di documento

LyX consente di creare un documento a partire da un modello, utilizzando la voce *New from template* del menù *File*. In pratica, si tratta soltanto di file LyX di esempio che si trovano in una posizione conveniente del file system, che l'utente comune non può modificare.

Automatismi

Quando il testo prevede una numerazione, come avviene con i titoli delle sezioni, i riferimenti in nota, i riferimenti alle pagine di un indice, tutto avviene in modo automatico. Tuttavia, si tratta di un compito demandato al sistema di composizione (LaTeX, DocBook o altro). L'inserimento di oggetti di questo genere nel testo, si ottiene con le voci del menù *Insert*. Per esempio, per inserire un indice generale, si deve selezionare la voce *Lists & TOC*, quindi *Table of contents*.

Riferimenti

- *The document processor*
  http://www.lyx.org

[1] **LyX** GNU GPL

Introduzione a HieroTeX

Installazione 709
Sesh 710
Utilizzare HieroTeX 710
Due modi di usare HieroTeX 711
Scrittura normale 711
Codifica di HieroTeX 714
Riferimenti 720

HieroTeX è un sistema per la composizione con caratteri geroglifici attraverso LaTeX. Si compone di una serie di file di stile e una serie di file di caratteri tipografici; inoltre fornisce alcuni programmi di servizio, in particolare Sesh, il cui scopo è quello di filtrare un file LaTeX per comporre le istruzioni corrette per la generazione di un testo in geroglifico.

Probabilmente non esiste alcun pacchetto già pronto per la propria distribuzione GNU/Linux e occorre fare da soli: sia l'installazione degli stili e dei caratteri, sia la compilazione di Sesh.

Lo scopo di questo capitolo è solo quello di mostrare come si usa HieroTeX. Chi scrive queste informazioni non ha alcuna preparazione su tale forma di scrittura: l'unica motivazione da cui è nato questo capitolo è la curiosità. È probabile che in queste pagine appaiono degli esempi senza senso nella lingua dell'antico Egitto, cosa di cui deve tenere conto il lettore.

HieroTeX può essere ottenuto dal sito gestito dal suo stesso autore, Serge Rosmorduc e precisamente dall'URI *http://www.iut.univ-paris8.fr/~rosmord/archives/* , prelevando i file corrispondenti ai modelli: '`egyptomf-*.tar.gz`', '`egyptopk-*.tar.gz`', e '`egyptouser-*.tar.gz`'.

Installazione

Dopo aver prelevato i tre file indicati all'inizio, si estrae il loro contenuto, così si ottiene la directory '`HieroTeX/`' a partire da quella corrente.

```
tar xzvf egypto…tar.gz
```

La prima cosa da fare è installare i caratteri tipografici e gli stili per TeX. Dal momento che ogni distribuzione GNU/Linux è organizzata a modo suo, per quanto riguarda TeX, bisogna fare una piccola ricerca per determinare dove sono stati collocati gli altri. Occorre cercare la posizione dei file '`*.mf`', '`*.tfm`' e '`*.sty`'. A titolo di esempio, potrebbe trattarsi delle directory '`/usr/share/texmf/fonts/source/`***pacchetto_tex***' per i file '`*.mf`', della directory '`/usr/share/texmf/fonts/tfm/`***pacchetto_tex***' per i file '`*.tfm`' e della directory '`/usr/share/texmf/tex/latex/`***pacchetto_tex***' per gli stili. In tal caso, si potrebbe procedere come viene mostrato di seguito.

```
$ su [Invio]

# mkdir /usr/share/texmf/fonts/source/hierotex [Invio]

# mkdir /usr/share/texmf/fonts/source/hierotex/mf [Invio]

# mkdir /usr/share/texmf/fonts/source/hierotex/auxmf [Invio]

# mkdir /usr/share/texmf/fonts/tfm/hierotex [Invio]

# mkdir /usr/share/texmf/tex/latex/hierotex [Invio]

# cd HieroTeX [Invio]

# cp Fonts/mf/* /usr/share/texmf/fonts/source/hierotex/mf
[Invio]
```

```
# cp Fonts/auxmf/* /usr/share/texmf/fonts/source/hierotex/auxmf
```
[*Invio*]

```
# cp Fonts/font/*.tfm /usr/share/texmf/fonts/tfm/hierotex
```
[*Invio*]

```
# cp TEX/*.sty /usr/share/texmf/tex/latex/hierotex
```
[*Invio*]

```
# cp TEX/*.fd /usr/share/texmf/tex/latex/hierotex
```
[*Invio*]

Successivamente, occorre ricostruire i file '`ls-R`' all'interno della struttura di LaTeX. Questo lo si può ottenere attraverso '**`texconfig`**', selezionando la voce '**`REHASH`**' dal menù principale.

```
# texconfig
```
[*Invio*]

Durante l'installazione dei caratteri e degli stili, occorre fare attenzione ai permessi delle directory e dei file: i file devono essere leggibili a tutti, mentre le directory, oltre a questo, devono essere anche attraversabili.

Sesh

Sesh è un programma molto semplice, il cui scopo è quello di preelaborare un sorgente LaTeX, scritto inserendo caratteri geroglifici, ma in modo semplificato. Il risultato è un file LaTeX corretto, che però sarebbe più difficile da scrivere.

Questo programma è indispensabile per lavorare bene con HieroTeX, per cui è necessario procedere alla sua compilazione. Nella documentazione originale, si indica la necessità di mettere mano al file '`HieroTeX/variable.mk`'; tuttavia, per la compilazione di Sesh, ciò non dovrebbe essere necessario. Per la compilazione si entra nella directory contenente i sorgenti.

```
$ cd HieroTeX/Seshnesu
```
[*Invio*]

```
$ make configure
```
[*Invio*]

```
$ make sesh
```
[*Invio*]

Se si avvia '**`make`**' senza argomenti, si ottiene semplicemente un promemoria delle opzioni disponibili.

La compilazione genera il file eseguibile '**`sesh`**', che può essere collocato dove si ritiene più opportuno, purché da lì possa essere utilizzato.

Utilizzare HieroTeX

Per poter scrivere dei simboli geroglifici attraverso HieroTeX, è necessario importare uno stile di questo sistema e utilizzare i comandi relativi. Prima di analizzare la sintassi e il comportamento dei comandi specifici di HieroTeX, è opportuno iniziare con un esempio banale, in modo da verificarne il funzionamento.

```
\documentclass{report}
\usepackage{hierLtx}
\begin{document}
\begin{center}
\hieroglyphe{F/35} = nfr
\end{center}
\end{document}
```

Supponendo che il file si chiami '`prova.tex`', la sua composizione avviene nel modo solito:

```
$ latex prova.tex
```
[*Invio*]

```
$ dvips -o prova.ps prova.dvi
```
[*Invio*]

La stessa cosa potrebbe essere ottenuta con un esempio leggermente differente:

```
\documentclass{report}
\usepackage{hiero}
\begin{document}
\begin{center}
\begin{hieroglyph}
    F35
\end{hieroglyph}
= nfr
\end{center}
\end{document}
```

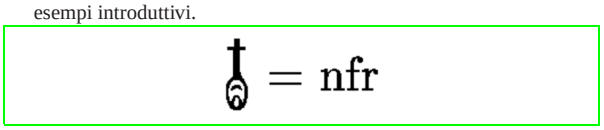
In tal caso, prima di dare in pasto questo file a LaTeX, occorre filtrarlo attraverso Sesh:


```
$ cat prova.tex | sesh > prova-1.tex
```
[*Invio*]

```
$ latex prova-1.tex
```
[*Invio*]

```
$ dvips -o prova.ps prova-1.dvi
```
[*Invio*]

Figura u95.3. Il risultato ingrandito della composizione dei due esempi introduttivi.



Due modi di usare HieroTeX

A seconda delle esigenze che si hanno, si può usare HieroTeX in due modi: incorporando lo stile '**`hierLtx`**' o lo stile '**`hiero`**'. Nel primo caso, per rappresentare i caratteri geroglifici si può usare solo il comando '**`\hieroglyphe{`**…**`}`**', mentre nel secondo si usa un ambiente: '**`\begin{hieroglyph}`**…**`\end{hieroglyph}`**'. Tuttavia, a seconda della situazione cambia il modo in cui i simboli geroglifici vanno annotati.

In generale, con il comando '**`\hieroglyphe{`**…**`}`**' si possono indicare i simboli nella forma '***lettera/numero***', per cui,

```
\hieroglyphe{F/35}
```

corrisponde al simbolo già mostrato nella figura u95.3. Al contrario, per fare la stessa cosa nell'altro modo, bisognerebbe scrivere:

```
\begin{hieroglyph}{\leavevmode \Hunh{\Aca F/35/}}\end{hieroglyph}
```

Tuttavia, disponendo dell'aiuto di Sesh, è sufficiente scrivere invece la sigla del geroglifico, nella forma '***letteranumero***' (senza la barra):

```
\begin{hieroglyph}F35\end{hieroglyph}
```

In generale, può essere conveniente utilizzare il primo metodo solo per scrivere poche cose, in modo tale da non dipendere da Sesh per annotare uno o due simboli; ma per fare qualcosa di più, è molto meglio scegliere il secondo stile utilizzando Sesh prima della composizione.

I simboli geroglifici devono poter essere raggruppati assieme stabilendo anche la sovrapposizione eventuale. Per entrambi gli stili di scrittura si possono usare il trattino singolo ('`-`') e i due punti ('`:`'), per ottenere rispettivamente la separazione orizzontale e la separazione verticale. Si osservino i due esempi seguenti che generano lo stesso risultato:

```
\hieroglyphe{M/17-X/1:N/35:N/5}
```

```
\begin{hieroglyph}M17-X1:N35:N5\end{hieroglyph}
```

Figura u95.9. '**`M17-X1:N35:N5`**'.

Nel caso particolare del comando '**`\hieroglyphe{`**…**`}`**', si possono raggruppare più segni tra parentesi graffe; volendo scrivere in modo più preciso quanto è già stato mostrato, si potrebbero riunire i tre simboli finali:

```
\hieroglyphe{M/17-{X/1:N/35:N/5}}
```

L'ambiente '**`hieroglyph`**' offre di più e questo viene descritto nella prossima sezione.

Scrittura normale

Per poter scrivere in maniera «decente» un testo con simboli geroglifici, occorre utilizzare la seconda modalità, quella che si avvale dell'aiuto di Sesh. A differenza del primo modo, i simboli possono essere indicati attraverso la sigla corrispondente, senza barre di separazione, oppure attraverso la loro traslitterazione, ammesso che esista. La codifica utilizzata deriva dal documento *Inventaire des signes hieroglyphiques en vue de leur saisie informatique*, citato alla

fine del capitolo e noto anche come *manuel de codage*, benché non sia perfettamente aderente a quel documento. A partire dalla figura u95.26 vengono elencati i codici disponibili con HieroTeX; tuttavia, dal momento che la qualità di queste immagini non è molto buona, conviene eventualmente fare riferimento alla tabella relativa contenuta nel documento *A LaTeXperiment of hieroglyphic typesetting*, sempre citato alla fine del capitolo.

I simboli, indicati attraverso la sigla standard, oppure la loro traslitterazione, possono essere separati nel modo già visto, attraverso il trattino e i due punti ('`-`', '`:`'), mentre il raggruppamento si fa attraverso l'uso delle parentesi tonde. Ma in questo ambiente sono possibili anche altri effetti, riepilogati in parte nella tabella u95.11. Inoltre, è possibile anche la scrittura incolonnata. Prima di illustrare in che modo è possibile ottenere l'incolonnamento, vengono mostrati alcuni esempi comuni, escluso il caso del raggruppamento che è già stato presentato.

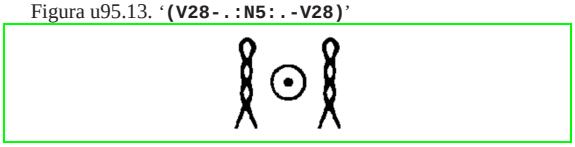
Tabella u95.11. Alcuni dei simboli speciali per la scrittura.

Codice	Risultato
`-`	Separa orizzontalmente.
`:`	Separa verticalmente.
`(…)`	Raggruppa.
`*`	Separa allo stesso livello.
`#`	Sovrascrive.
`-=`	Conclusione grammaticale.
`:=`	Conclusione grammaticale.
`..`	Spazio.
`.`	Mezzo spazio.
`\`	Ruota orizzontalmente il simbolo che lo precedente.
`\s`***n***	Riduce la dimensione del simbolo che lo precede di ***n*** volte.
`<-…->`	Delimita all'interno di un cartiglio.
`<S-…->`	Delimita all'interno di un «serekh».
`<Sb-…->`	Delimita mostrando solo l'inizio di un serekh.
`<Sm-…->`	Delimita mostrando solo la parte centrale di un serekh.
`<Se-…->`	Delimita mostrando solo la parte finale di un serekh.
`<H-…->`	Delimita all'interno di un segno «ḥwt».
`-#-…-#-`	Ombreggiatura dei simboli contenuti.
`+l…+s`	Delimita del testo normale (LaTeX).
`|…-`	Pone il testo normale elevato all'esponente.
`\!`	Avvicinamento tra i simboli.
`-!` `:!`	Chiude una colonna.

- **Spaziatura**

```
\begin{hieroglyph}
    (V28-.:N5:.-V28)
\end{hieroglyph}
```

Figura u95.13. '**`(V28-.:N5:.-V28)`**'



La figura u95.13 mostra il risultato della composizione. Si osservi l'uso del punto singolo, come richiesta esplicita di un piccolo spazio, prima e dopo il simbolo N5. Senza questa spaziatura, il simbolo apparirebbe troppo basso.

- **Rotazione orizzontale**

```
\begin{hieroglyph}
    (A1-A1\)
\end{hieroglyph}
```

figura u95.15. '**`(A1-A1\)`**'

La figura u95.15 mostra il risultato della composizione. L'inversione del secondo simbolo è stato ottenuto aggiungendo in coda una barra obliqua inversa ('\').

- **Cartiglio**

```
\begin{hieroglyph}
    <-(M17-X1:N35:N5)-(G25-\!\!Aa1:.):N35->
\end{hieroglyph}
```

Figura u95.17. '**`<-(M17-X1:N35:N5)-(G25-\!\!Aa1:.):N35->`**'

La figura u95.17 mostra il risultato della composizione. Il trattino utilizzato all'interno dei simboli '**<**' e '**>**' serve solo a evitare ambiguità con altri comandi particolari, ovvero con altri tipi di cornici diverse dal cartiglio.

- **Avvicinamento**

```
\begin{hieroglyph}
    (G39-\!\!N5:.)
\end{hieroglyph}
```

Figura u95.19. '**`(G39-\!\!N5:.)`**'

```
\begin{hieroglyph}
    (I10:\!\!X1:N17)
\end{hieroglyph}
```

Figura u95.21. '**`(I10:\!\!X1:N17)`**'

Le figure u95.19 e u95.21 mostrano rispettivamente i due esempi, dove nel primo caso c'è un avvicinamento di simboli in modo orizzontale, mentre nel secondo si ha un avvicinamento in modo verticale.

Vale la pena di annotare che uno o più spazi rappresentano la fine di una parola. Gli spazi vanno messi prima dei simboli di separazione (il trattino e i due punti) e questo, tra le altre cose, facilita l'incolonnamento del testo nel sorgente LaTeX.

Per incolonnare i simboli geroglifici, si utilizza il comando seguente,

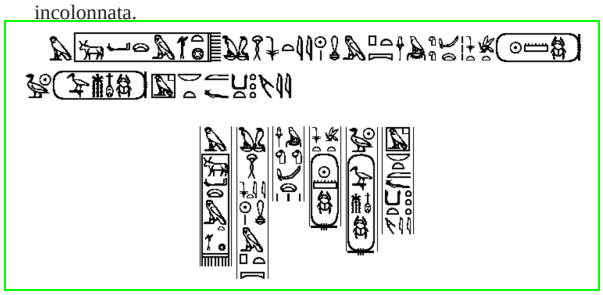
```
\EnColonne[dimensione\Htm]{…}
```

dove all'interno delle parentesi graffe va dichiarato l'ambiente '**`hieroglyph`**'. Viene mostrato un esempio abbastanza complesso, tratto dalla documentazione di HieroTeX. Viene abbinato lo stesso testo, prima in forma orizzontale, poi in forma verticale:

```
\begin{hieroglyph}
    G5 <S E1 D40 xa m R19*(t:niwt) > nbty wAH sw*t-i-i (ra:Z1)*mi m p*t:pt
    sxm*G8*(F9:F9) (Dsr:xa)*Z3 (sw:t)*(bit:t) <ra-mn-xpr> zA*\!\!(ra:.)
    <G26-ms*nfr-xpr> O10 nb:t M:f (kA:t)*(N33:N33:N33) mr*i*i
\end{hieroglyph}
```

```
\begin{center}
    \EnColonne[1.2\Htm]{
        \begin{hieroglyph}
            G5 <S E1 D40 xa m R19*(t\s1:niwt\s1) >-!
            nbty wAH sw*\!t\s1*\!i*i (ra:Z1)*mi m (p*t:pt)-!
            sxm*G8 F9*F9  Dsr-xa-Z2-!
            (sw:t)*(bit:t) <ra-mn-xpr>-! zA*\!\!\!(ra:.)
            <-G26-ms*nfr-xpr->-! O10 nb-t M-f (kA:t)*(N33:N33:N33) mr*i*i
        \end{hieroglyph}
    }
\end{center}
```

Figura u95.24. Esempio di una composizione normale e incolonnata.



La figura u95.24 mostra il risultato di questa composizione. Si osservi che in questo caso, quando possibile, è stata usata la codifica corrispondente alla traslitterazione invece del nome nella solita forma '***lettera numero***'.

Codifica di HieroTeX

Nelle prossime pagine viene mostrato un elenco di simboli geroglifici e la loro codifica corrispondente. Si può notare che a volte sono disponibili più forme diverse per la codifica; di solito, da quanto propone l'autore di HieroTeX, si tende a preferire quella che si avvicina di più alla traslitterazione del simbolo. La prima figura, u0.3, mostra l'elenco dei simboli alfabetici fondamentali; le altre mostrano tutti i simboli disponibili.

Figura u95.25. Codifica alfabetica fondamentale.

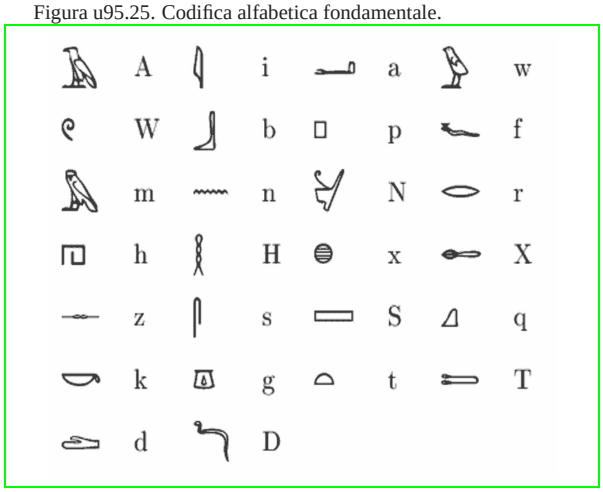


Figura u95.26. Codifica usata da HieroTeX.

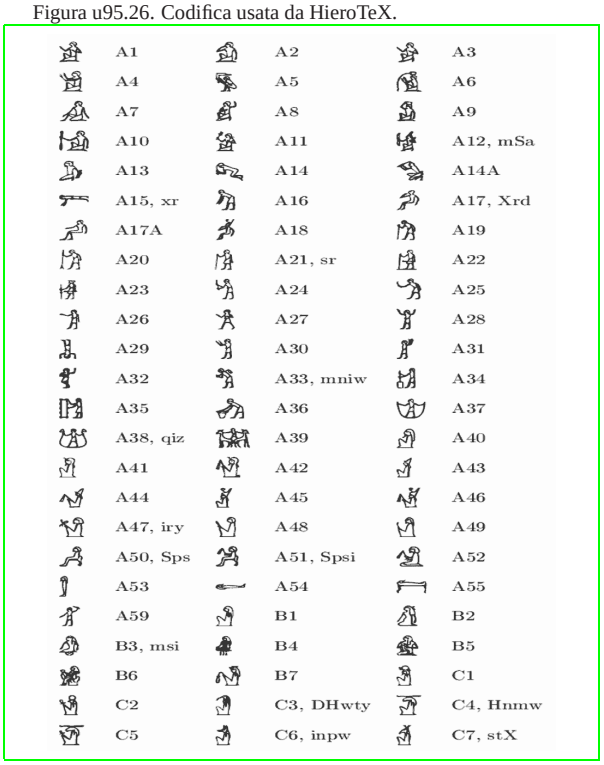


Figura u95.27. Codifica usata da HieroTeX.

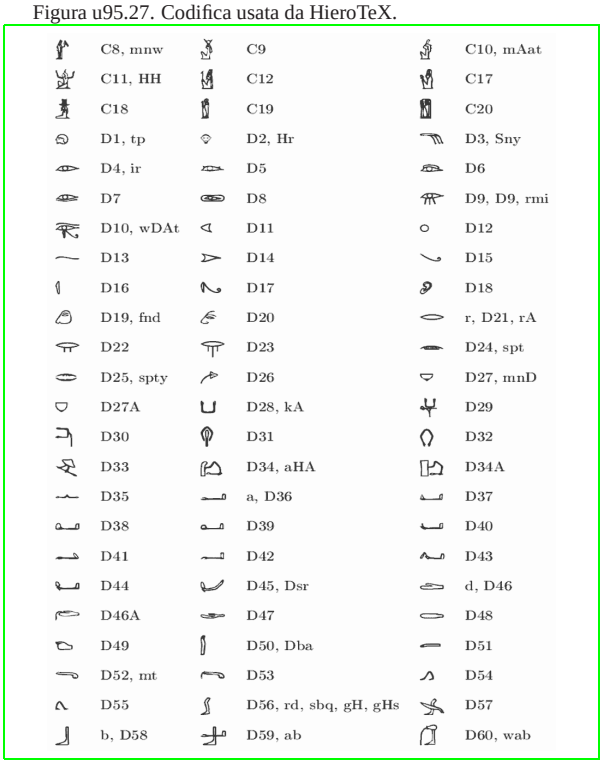


Figura u95.28. Codifica usata da HieroTeX.

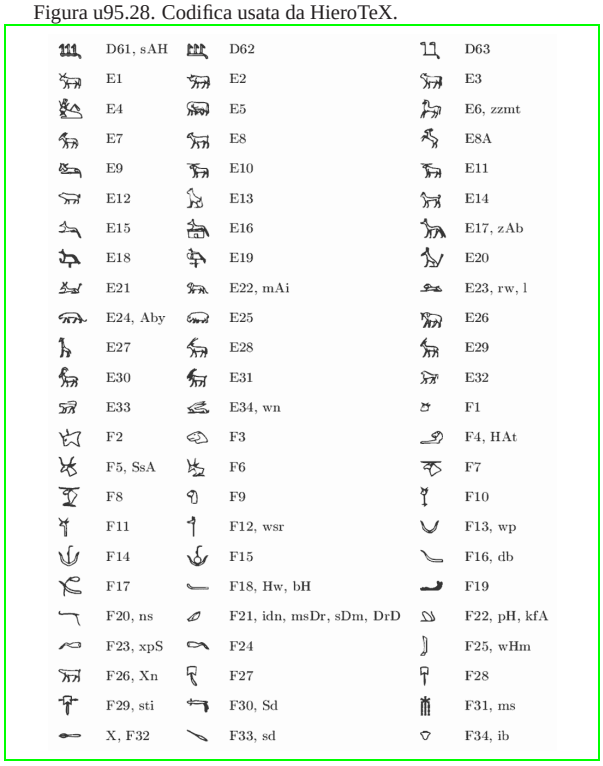


Figura u95.29. Codifica usata da HieroTeX.

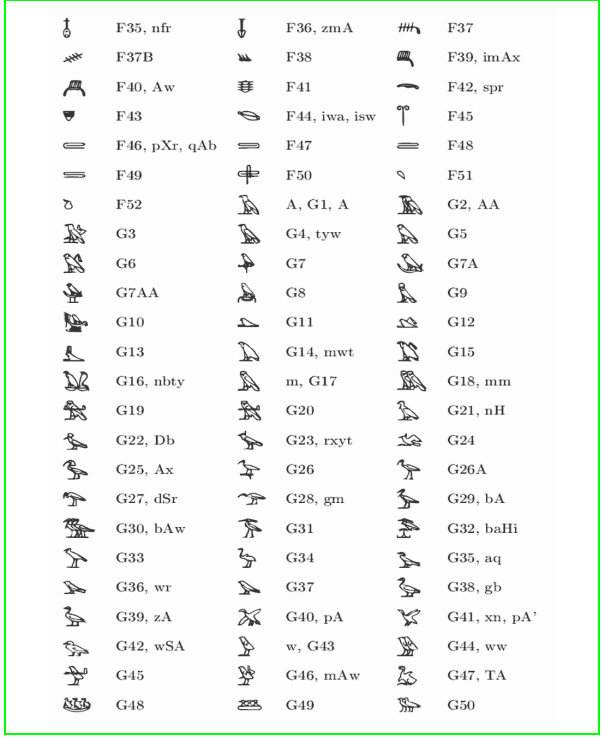


Figura u95.30. Codifica usata da HieroTeX.

Figura u95.31. Codifica usata da HieroTeX.

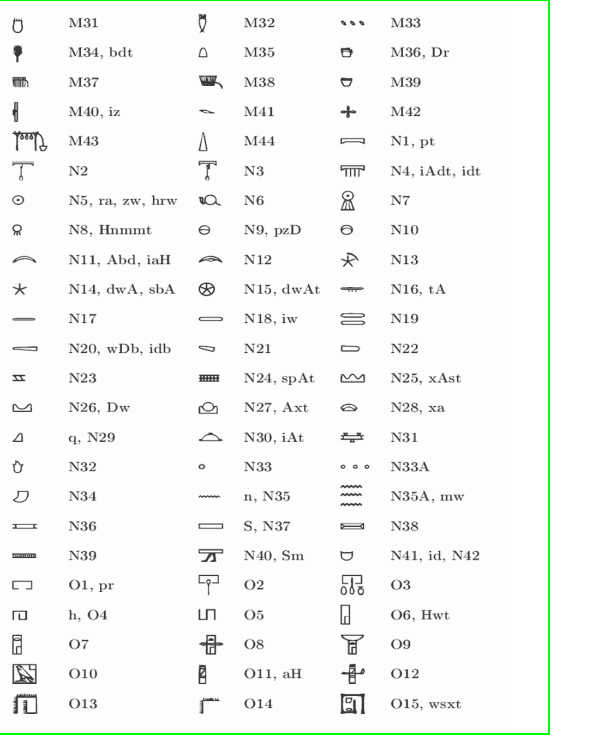


Figura u95.32. Codifica usata da HieroTeX.

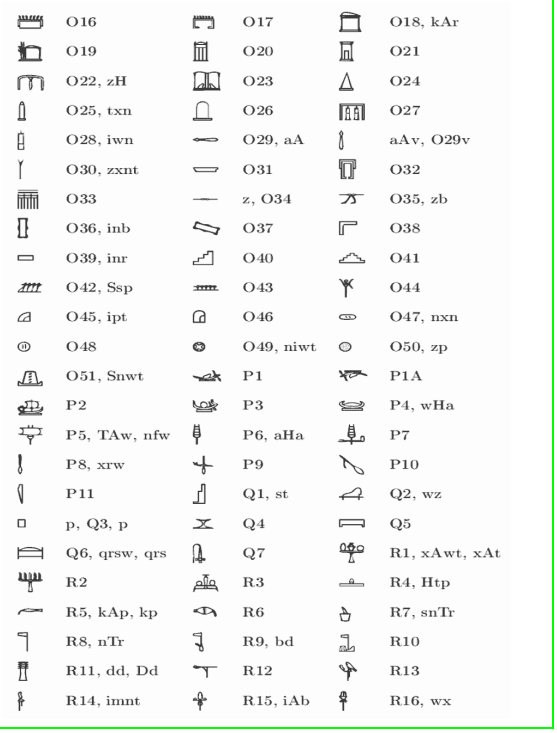


Figura u95.33. Codifica usata da HieroTeX.

Figura u95.34. Codifica usata da HieroTeX.

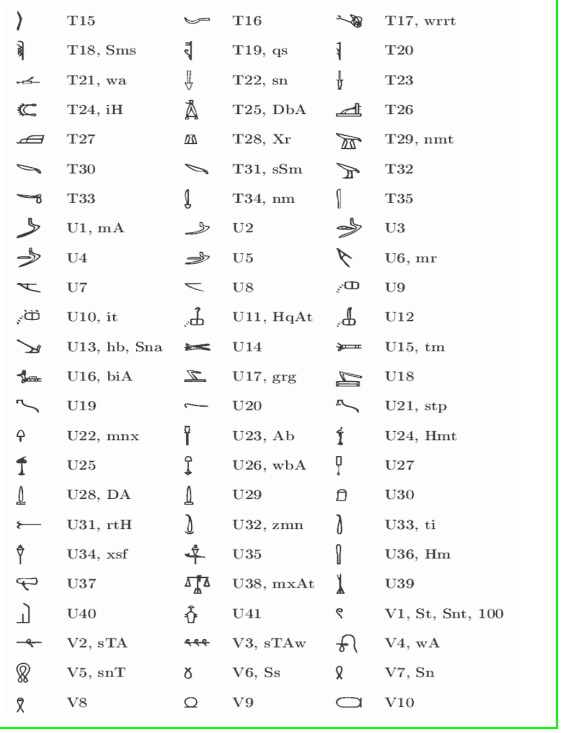
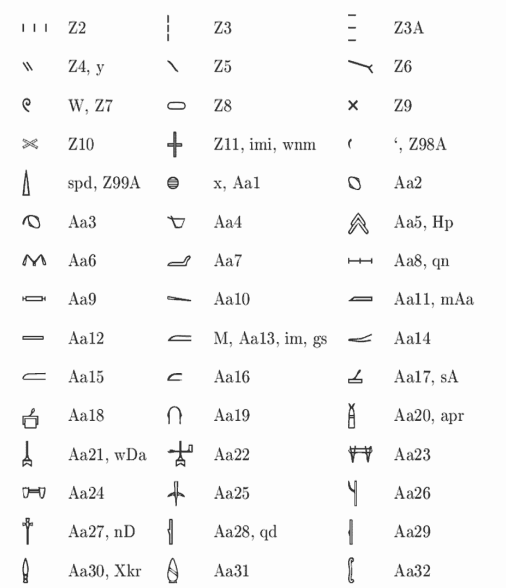


Figura u95.35. Codifica usata da HieroTeX.

Figura u95.36. Codifica usata da HieroTeX.



Riferimenti

- Serge Rosmorduc, *A Short Introduction to Hieroglyphs* *http://www.iut.univ-paris8.fr/~rosmord/Intro/Intro.html*
- Jan Buurman, Nicolas Grimal, Michael Hainsworth, Jochen Hallof, Dirk Van Der Plas, *Inventaire des signes hieroglyphiques en vue de leur saisie informatique*, Mémoires de l’Académie des Inscriptions et Belle Lettres, Institut de France, Paris, 1988

Trasformazione in altri formati

DLH: trasforma LaTeX in HTML 721
Help2man: genera una pagina di manuale dalle informazioni fornite dal programma 722
Pstotext: estrae il testo da un file PostScript o PDF 723
Mswordview 723
Catdoc 723
Antiword 724
xlHTML 725

Spesso ci si trova di fronte alla necessità o all’utilità di trasformare un documento scritto in un certo modo, per esempio in LaTeX, in qualcosa di diverso, per esempio in HTML. In generale, queste cose andrebbero pianificate prima, per decidere lo stile del documento in base alle forme in cui questo deve poi concretizzarsi. Meglio ancora sarebbe l’utilizzo di strumenti appositi, di solito SGML/XML, pensati in anticipo per la produzione di documentazione in formati differenti.

Questo capitolo serve a raccogliere la descrizione di strumenti che possono aiutare a trasformare un documento realizzato con sistemi di composizione tradizionale, pensati principalmente per la stampa su carta, e viceversa.

Non ci si possono fare illusioni: gli strumenti di questo tipo non funzionano sempre, ma solo quando le caratteristiche del sorgente lo consentono.

DLH: trasforma LaTeX in HTML

DLH [1] è uno strumento relativamente semplice per la conversione di sorgenti LaTeX in HTML. La trasformazione avviene con successo solo quando si tratta di un sorgente LaTeX in cui non si usano ambienti matematici e soprattutto non si usano comandi particolarmente sofisticati (ciò inteso dal punto di vista di DLH).

DLH utilizza un insieme personalizzato di stili LaTeX, collocato normalmente nella directory ‘`/usr/share/dlh/inputs/dlh/`’. Si tratta dei soliti ‘`article.sty`’, ‘`epsfig.sty`’ e altri, ma il contenuto di questi file è ridotto rispetto a quelli equivalenti di LaTeX. Se nel sorgente LaTeX si utilizzano altri stili particolari occorrerebbe creare un file corrispondente anche in questa directory, cercando di adattarlo a DLH (cosa che potrebbe risultare difficile, dal momento che bisogna ragionare in termini di TeX limitato secondo le possibilità di DLH).

Il programma eseguibile è ‘**`dlh`**’ che accetta l’indicazione di alcune opzioni e in particolare un elenco di file LaTeX:

`dlh` [*opzioni*] *file_latex*...

In corrispondenza dei file indicati come argomento vengono create altrettante directory contenenti una serie di file HTML che rappresentano il risultato della trasformazione (a partire da ‘`index.html`’ che normalmente è un collegamento simbolico al primo di questi file).

DLH utilizza una serie di icone per rappresentare i pulsanti per lo scorrimento del documento secondo la sua struttura. I file di queste icone si trovano normalmente nella directory ‘`/usr/share/dlh/icons/`’ e andrebbero copiati nella directory ‘`../icons/`’, rispetto a quella in cui si trovano i file HTML.

Tabella u96.1. Alcune opzioni.

Opzione	Descrizione
`-f` `--force`	Questa opzione serve a creare tutti i file che compongono il documento, in particolare le immagini. Ciò può creare un rallentamento nel funzionamento di DLH, ma in generale serve a garantire un risultato più sicuro.
`-i` ***uri*** `--icon-dir=`***uri***	Permette di definire esplicitamente la collocazione dei file che rappresentano le icone utilizzate da DLH per rappresentare i pulsanti per lo scorrimento del documento.

Segue la descrizione di alcuni esempi.

- `$` **`dlh prova.tex`** [*Invio*]

  Crea la directory '`./prova/`' e al suo interno inserisce una serie di file HTML che riproducono il documento '`prova.tex`'. In questo caso, i file HTML fanno uso delle icone che si trovano nella directory '`./icons/`', relativa al nodo di rete in cui si trovano.

- `$` **`dlh -f prova.tex`** [*Invio*]

  Come nell'esempio precedente, ma viene forzata la creazione di tutti i file, nel caso ce ne fosse bisogno.

- `$` **`dlh -i icone prova.tex`** [*Invio*]

  Come nel primo esempio, con la differenza che i file delle icone devono trovarsi nella directory '`./prova/icone/`'.

Help2man: genera una pagina di manuale dalle informazioni fornite dal programma

Help2man [2] è un programma in grado di generare una pagina di manuale a partire dalle informazioni che restituisce un altro programma attraverso le opzioni '**`--help`**' e '**`--version`**'.

Help2man è predisposto principalmente per gestire convenientemente il risultato generato da un programma che segue le convenzioni GNU (ovvero della Free Software Foundation).

`help2man` [***opzioni***] ***programma_eseguibile***

Lo schema sintattico permette di vedere che si tratta dell'eseguibile '**`help2man`**', che oltre alle opzioni eventuali richiede l'indicazione di un programma da avviare con le opzioni '**`--help`**' e '**`--version`**' per ottenere le informazioni necessarie. In modo predefinito, il risultato viene emesso attraverso lo standard output.

Tabella u96.2. Alcune opzioni.

Opzione	Descrizione
`-o` ***file*** `--output=`***file***	Permette di definire il nome del file da generare, evitando così di emettere il risultato attraverso lo standard output.
`-s` ***n_sezione*** `--section=`***n_sezione***	Permette di specificare il numero della sezione della pagina di manuale.

Segue la descrizione di alcuni esempi.

- `$` **`help2man ls > ls.1`** [*Invio*]

  Genera il file '`ls.1`', contenente la pagina di manuale di '**`ls`**'.

- `$` **`help2man -o ls.1 ls`** [*Invio*]

  Esattamente come nell'esempio precedente, utilizzando esplicitamente l'opzione '**`-o`**'.

Pstotext: estrae il testo da un file PostScript o PDF

Pstotext [3] è un programma molto semplice per l'estrazione del testo contenuto all'interno di un file PostScript o PDF, per mezzo di Ghostscript.

`pstotext` [***opzioni***] ***file***

Tutto il lavoro viene svolto dall'eseguibile '**`pstotext`**'. Il risultato dell'elaborazione viene emesso attraverso lo standard output, a meno che sia stato stabilito diversamente con le opzioni.

Tabella u96.3. Alcune opzioni.

Opzione	Descrizione
`-cork`	Specifica che il file PostScript utilizza la codifica «cork», ovvero ciò che si ottiene da Dvips quando questo converte file DVI generati da TeX con la codifica T1.
`-landscape` `-landscapeOther`	Queste due opzioni indicano che il testo è ruotato a 90 gradi in un senso, oppure nell'altro.
`-portrait`	In questo caso si intende che il testo scorre come di consueto, su un foglio orientato in modo verticale.
`-output` ***file***	Consente di indicare il file di testo da generare, senza bisogno di ridirigere lo standard output.

Mswordview

Mswordview [4] è un programma il cui scopo è quello di convertire file di MS-Word in HTML. La conversione non può essere perfetta, ma il progetto è condotto con impegno e i risultati che dà questo programma sono buoni.

L'eseguibile di questo programma corrisponde a '**`mswordview`**' e la sintassi per il suo utilizzo si può schematizzare secondo il modello seguente:

`mswordview` [***opzioni***] ***file_doc***

Mswordview è in grado di convertire solo un file alla volta, precisamente quello che viene indicato alla fine degli argomenti. Se non viene richiesto qualcosa di particolare attraverso le opzioni, Mswordview tenta di creare un file con lo stesso nome di quello che viene convertito, con l'aggiunta dell'estensione '`.html`'. Inoltre, se il file contiene delle immagini incorporate, queste vengono trasferite su file esterni.

Tabella u96.4. Alcune opzioni.

Opzione	Descrizione
`-o` ***file_html*** `--outputfile` ***file_html***	Permette di indicare esplicitamente il file HTML che si vuole generare.
`-g` ***file_errori*** `--errorfile` ***file_errori***	Permette di annotare gli errori incontrati durante la conversione nel file indicato.

Catdoc

Catdoc [5] è un programma molto semplice, che si sostituisce idealmente a '**`cat`**' quando si tratta di visualizzare il contenuto di file scritti in formato MS-Word. Il suo funzionamento è intuitivo e in generale non servono opzioni: il file indicato come argomento, o fornito attraverso lo standard input, viene emesso dallo standard output dopo una conversione in formato testo. Se il file originale contiene in realtà solo testo puro, non avviene alcuna conversione.

`catdoc` [***opzioni***] ***file_doc***
`catdoc` [***opzioni***] `<` ***file_doc***

Tabella u96.5. Alcune opzioni.

Opzione	Descrizione
`-b`	Cerca di elaborare anche file MS-Word che apparentemente non lo sono, a causa di una firma iniziale errata.
`-m`*n*	Specifica il margine destro del testo ottenuto. Il margine predefinito è a colonna 72. Si osservi che l'opzione '**`-m0`**' equivale a '**`-w`**'.
`-w`	Specifica il margine destro del testo ottenuto di lunghezza indefinita, in modo da ottenere che i paragrafi occupino una riga intera.
`-v`	Genera alcune informazioni diagnostiche prima del testo trasformato.

Per quanto semplice possa essere questo programma, è prevista una configurazione, composta dal file '`/etc/catdocrc`' per il sistema e dai file '`~/.catdocrc`' per gli utenti. Senza entrare nel dettaglio delle direttive di configurazione, è il caso di descrivere quella che rappresenta l'impostazione comune:

```
charset_path=/usr/lib/catdoc
map_path=/usr/lib/catdoc
source_charset=cp1252
target_charset=UTF-8
unknown_char='?'
```

Come si può intuire, le direttive '**`charset_path`**' e '**`map_path`**' servono a indicare la collocazione di file utilizzati da Catdoc per la conversione. La direttiva '**`source_charset`**' permette di stabilire la codifica predefinita del file sorgente, quando questo non appare utilizzare la UTF-16. La direttiva '**`target_charset`**' permette di definire la codifica da usare per il testo generato; come si vede nell'esempio viene usata la codifica UTF-8. Infine, è possibile stabilire in che modo mostrare i caratteri che non possono essere rappresentati, attraverso la direttiva '**`unknown_char`**', che in questo caso usa il punto interrogativo.

Segue la descrizione di alcuni esempi.

- `$` **`catdoc pippo.doc | less`** [*Invio*]

  Visualizza il contenuto del file '`pippo.doc`', con l'aiuto di '**`less`**' per scorrerlo.

- `$` **`catdoc pippo.doc > pippo.txt`** [*Invio*]

  Genera il file '`pippo.txt`' a partire da '`pippo.doc`'.

Antiword

Antiword [6] è un programma molto semplice per convertire file dal formato MS-Word in testo puro e semplice, oppure in PostScript , estrapolando anche le immagini. Il suo funzionamento è intuitivo e in generale non servono opzioni: il file indicato come argomento, viene emesso attraverso lo standard output dopo la conversione.

`antiword` [***opzioni***] ***file_doc***...

Tabella u96.7. Alcune opzioni.

Opzione	Descrizione
`-t`	Genera una conversione in formato testo puro e semplice. L'uso di questa opzione è implicito.
`-w` ***n_colonne***	Permette di specificare, nell'ambito di una conversione in formato testo, l'ampiezza del testo in caratteri. Se si utilizza il valore zero, si ottiene ogni paragrafo in una sola riga.
`-m` ***file_mappa***	Consente di indicare la codifica del file di testo che si vuole ottenere.

Opzione	Descrizione
`-p` ***dimensioni_carta***	L'utilizzo di questa opzione richiede implicitamente la conversione in formato PostScript, mentre in condizioni normali si ottiene un testo puro e semplice. L'argomento dell'opzione stabilisce la dimensione della carta e può trattarsi delle parole chiave seguenti, con il significato intuitivo che hanno: '**`10x14`**', '**`a3`**', '**`a4`**', '**`a5`**', '**`b4`**', '**`b5`**', '**`executive`**', '**`folio`**', '**`legal`**', '**`letter`**', '**`note`**', '**`note`**', '**`quarto`**', '**`statement`**', '**`tabloid`**'.
`-L`	Nell'ambito di una conversione in PostScript, indica un orientamento orizzontale del foglio.
`-i` ***livello_di_visualizzazione_immagini***	Consente di specificare cosa fare delle immagini che fossero eventualmente contenute nel file di partenza. L'argomento è un numero.
`-i 0`	Genera un file compatibile con Ghostscript, ma non adatto a stampanti PostScript comuni. Tuttavia, in condizioni normali, se si arriva alla stampa, si passa generalmente per Ghostscript, per cui questo valore è quello che può essere adatto.
`-i 1`	Non estrapola le immagini.
`-i 2`	PostScript livello 2.
`-i 3`	PostScript livello 3.
`-s`	Include anche il testo nascosto, indicato come tale nel file originale.

Segue la descrizione di alcuni esempi.

- `$` **`antiword pippo.doc | less`** [*Invio*]

  Visualizza il contenuto del file '`pippo.doc`', con l'aiuto di '**`less`**' per scorrerlo.

- `$` **`antiword -p a4 pippo.doc > pippo.ps`** [*Invio*]

  Genera il file '`pippo.ps`' (PostScript, A4) a partire da '`pippo.doc`'.

xlHTML

xlHTML[7] è un programma per convertire file dal formato MS-Excel in HTML, come suggerisce il nome, oppure in testo puro. Se non si usano le opzioni, si ottiene un file HTML, contenente una tabella con ciò che appare nel foglio elettronico indicato nella riga di comando, emesso attraverso lo standard output:

`xlhtml` [***opzioni***] ***file_xls*** `>` ***file***

Tabella u96.8. Alcune opzioni.

Opzione	Descrizione
`-fw`	In condizioni normali, le celle che contengono delle espressioni vengono valutate e ne viene mostrato solo il risultato, con una nota sulla possibilità che il valore mostrato non sia preciso. Per fare sparire questa nota si usa l'opzione '**`-fw`**'.
`-c`	Fa in modo che la tabella contenente i dati del foglio elettronico appaia centrata nel corpo della pagina HTML.
`-asc`	Genera un risultato in formato testo puro, se però si abbina anche una delle opzioni che iniziano per '**`-x`**'.

Opzione	Descrizione
`-csv`	Genera un risultato in formato testo, dove i campi sono delimitati da apici doppi e separati da una virgola. Anche in questo caso, l'opzione funziona solo in abbinamento con una delle opzioni che iniziano per '`-x`'.
`-xp:`***n***	Converte solo la pagina ***n***, contando a cominciare da zero.
`-xc:`***m*`-`*n***	Converte solo le colonne da ***m*** a ***n***, contando a cominciare da zero.
`-xr:`***m*`-`*n***	Converte solo le colonne da ***m*** a ***n***, contando a cominciare da zero.

Per comprendere le possibilità di xlHTML viene mostrato un solo esempio, di un foglio elettronico realizzato con Gnumeric, salvando in formato XLS. Due figure mostrano il contenuto del foglio, sia nel suo aspetto finale, sia nel contenuto effettivo delle celle.

Figura u96.9. Il foglio elettronico di esempio, nel suo aspetto finale.

	A	B	C	D	E
1	Straordinario				
2	Data	dalle ore	alle ore	durata	
3	15/01/2005	19:30	21:30	2:00	
4	16/01/2005	19:30	21:30	2:00	
5	17/01/2005	19:30	21:30	2:00	
6			Totale straordinario:	6:00	
7					
8					

Figura u96.10. Il foglio elettronico di esempio con le espressioni contenute nelle celle.

	A	B	C	D	E
1	Straordinario				
2	Data	dalle ore	alle ore	durata	
3	=date(2005;1;15)	=time(19;30;0)	=time(21;30;0)	=C3-B3	
4	=date(2005;1;16)	=time(19;30;0)	=time(21;30;0)	=C4-B4	
5	=date(2005;1;17)	=time(19;30;0)	=time(21;30;0)	=C5-B5	
6			Totale straordinario:	=sum(D3:D5)	
7					

Supponendo che il file si chiami '`esempio.xls`', si può procedere con il comando seguente per generare il file '`esempio.html`':

```
$ xlhtml -fw esempio.xls > esempio.html [ Invio ]
```

Il file che si ottiene dovrebbe avere l'aspetto seguente; si osservi che le date non sono state rappresentate in modo corretto:

Sheet1

Straordinario			
Data	dalle ore	alle ore	durata
38367 *	19:30	21:30	2:00
38368 *	19:30	21:30	2:00
38369 *	19:30	21:30	2:00
	Totale straordinario:		6:00

Spreadsheet's Author: Unknown
Last Updated with Excel 97
* This cell's format is not supported.

Created with xlhtml 0.5.1

[1] **DLH** GNU GPL

[2] **Help2man** GNU GPL

[3] **Pstotext** licenza speciale

[4] **Mswordview** GNU GPL + alcuni file con licenza speciale

[5] **catdoc** GNU GPL

[6] **Antiword** GNU GPL

[7] **xlHTML** GNU GPL

Operatori logici e porte logiche 729
Operatori unari 730
Connettivo AND 730
Connettivo OR 731
Connettivo XOR 731
Ordine di precedenza 731
Valori irrilevanti nelle tabelle di verità 732
Equivalenze 732
Somma dei prodotti 733
Mappe di Karnaugh 734
Mappe di Karnaugh con la funzione XOR 737
Circuiti combinatori 741
Decodificatore 742
Demultiplatore 743
Multiplatore 744
Demultiplatori e multiplatori in parallelo 745
Codificatore binario 745
Codificatore di priorità 746
Unità logiche 747
Scorrimento 748
Addizionatore 750
Sottrazione 752
Somma e sottrazione assieme 753
Riporto anticipato 754
Complemento a due 756
Moltiplicazione 757
Divisione 761
Comparazione 763
Costruzione di un'unità aritmetico-logica 765
Indicatori 765
Troncamento di un valore in base al rango 766
Inversione di segno 767
Somma e sottrazione 768
Scorrimento e rotazione 768
Comparazione di valori 770
Moltiplicazione 772
Divisione 774
Unità logica 776
ALU completa 777
Unità aritmetico-logica e registri espandibili 779
Unità aritmetico-logica a bit singolo 779
Registri 781
Flip-flop 783
Ritardo di propagazione 783
Tabelle di verità 784
Flip-flop SR elementare 784
Interruttori senza rimbalzi 787
Flip-flop SR con gli ingressi controllati 787
Flip-flop SR sincrono «edge triggered» 788
Tempo: *setup*/*hold* e *recovery*/*removal* 789
Flip-flop D 790
Flip-flop T 791
Flip-flop JK 792

Registri 795
Registri semplici 795
Registri a scorrimento 797
Contatori asincroni con flip-flop T 798
Contatori sincroni con flip-flop T 799
Contatori sincroni con flip-flop D 799
Contatori sincroni con caricamento parallelo 800
Bus e unità di controllo 803
Bus con il buffer a tre stati 803
Unità di controllo del bus 804
Microcodice 806
Tkgate 809
File PostScript ed EPS 809
Rappresentazione di multiplatori, demultiplatori e codificatori 809
Clock 811
Riferimenti 813

Operatori logici e porte logiche

Operatori unari 730
Connettivo AND 730
Connettivo OR 731
Connettivo XOR 731
Ordine di precedenza 731
Valori irrilevanti nelle tabelle di verità 732
Equivalenze 732
Somma dei prodotti 733
Mappe di Karnaugh 734
Mappe di Karnaugh con la funzione XOR 737

La logica formale studia le proposizioni dichiarative, dove per proposizione si intende l'insieme di soggetto e verbo. È una ***proposizione dichiarativa*** quella proposizione nei confronti della quale è possibile stabilire se è vera o se è falsa. *Vero* o *Falso* sono gli unici valori che può assumere una proposizione dichiarativa. La proposizione che non si può suddividere in altre proposizioni, si dice essere elementare.

Il valore di una proposizione dichiarativa (*Vero* o *Falso*) può essere espresso in vari modi, a seconda del contesto. Generalmente, si attribuisce alla cifra numerica uno il significato di *Vero*, mentre a zero si attribuisce il valore *Falso*, così come nell'applicazione elettronica si utilizza un livello di tensione vicino a quello di alimentazione per rappresentare *Vero* e un valore di tensione vicino a zero per rappresentare *Falso*.

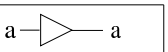
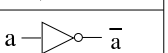
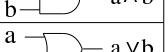
Falso	0	0 V	⏚
Vero	1	tensione di alimentazione	$\top_{Vcc,\,Vdd}$

La variabile che può assumere solo il valore risultante da una proposizione dichiarativa, è una ***variabile logica***. Un'***espressione logica*** o una ***funzione logica*** è quella che produce un risultato *Vero* o *Falso*. L'espressione o funzione logica può essere costituita da proposizioni dichiarative, da valori costanti (espressi secondo la forma prevista per rappresentare *Vero* o *Falso*) e da variabili logiche. Per connettere o comunque per intervenire nei valori delle varie componenti dell'espressione, si utilizzano degli operatori che rappresentano delle funzioni elementari.

Si distinguono generalmente gli operatori logici in «unari» e in «connettivi logici», per distinguere se intervengono in un solo valore logico, oppure su due o più valori logici. Gli operatori logici si possono vedere come delle scatoline, aventi uno o più ingressi, ma con una sola uscita: in tal caso, si chiamano ***porte logiche***.

Gli operatori logici hanno forme differenti di rappresentazione, in base al contesto e in base alle limitazioni tipografiche a cui si deve sottostare. Pertanto, ogni volta che si legge un documento che tratta questo genere di argomenti, è necessario inizialmente comprendere qual è la simbologia adottata, tenendo conto che anche all'interno dello stesso testo si possono alternare simbologie differenti.

Tabella u97.2. Alcune notazioni alternative degli operatori logici comuni, associate alla simbologia delle porte logiche corrispondenti.

a	a	a	a	
a	NOT a	a’	$\overline{a}$	
a b	a AND b	a · b	a ∧ b	
a b	a OR b	a + b	a ∨ b	
a b	a XOR b		a ⊕ b	

Per definire il significato degli operatori logici si utilizzano delle tabelle di verità, mettendo a confronto le variabili in ingresso con l’esito finale.

Operatori unari

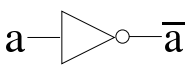
Gli operatori logici unari ottengono in ingresso un solo valore logico; sono disponibili l’invertitore logico (NOT) e il non-invertitore logico.

L’invertitore logico è l’operatore unario che inverte il valore logico ricevuto in ingresso: se in ingresso riceve il valore *Vero* (1), in uscita genera il valore *Falso* (0); se in ingresso riceve il valore *Falso* (0), in uscita genera il valore *Vero* (1).

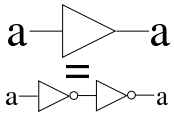
A titolo di esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia», l’espressione logica «NOT A» è equivalente alla proposizione dichiarativa «Antonio non mangia».

Il non-invertitore logico è l’operatore unario che presenta in uscita lo stesso valore ricevuto in ingresso. Il nome che viene dato a questo tipo di operatore allude alla presenza ipotetica di due negazioni consecutive che si eliminano a vicenda. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia», l’espressione logica «NOT A» è equivalente alla proposizione dichiarativa «Antonio non mangia», ma nello stesso modo, «NOT (NOT A)» è equivalente alla proposizione originale: «Antonio mangia».

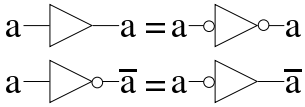
a	$\overline{a}$
0	1
1	0



a	a
0	0
1	1



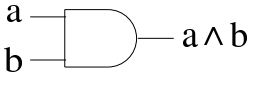
Dagli esempi mostrati si dovrebbe intendere il fatto che un piccolo cerchio rappresenta un’inversione logica, e si può collocare all’ingresso o all’uscita di una funzione logica rappresentata graficamente. Pertanto, valgono le equivalenze seguenti:



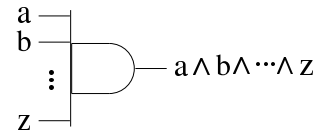
Connettivo AND

I connettivi logici sono gli operatori che utilizzano due ingressi. Il connettivo AND restituisce il valore *Vero* solo se entrambi i valori in ingresso sono pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero legge», l’espressione «A AND B» equivale alla proposizione «Antonio mangia e Piero legge».

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1



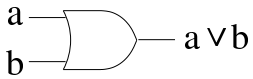
La porta logica AND, quando ha più di due ingressi, esprime l’equivalente di un’espressione in cui tutte le variabili in ingresso sono collegate dall’operatore AND:



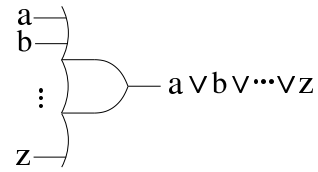
Connettivo OR

Il connettivo OR restituisce il valore *Vero* se almeno uno dei due ingressi dispone di un valore pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero legge», l’espressione «A OR B» equivale alla proposizione «Antonio mangia e/o Piero legge».

a	b	a∨b
0	0	0
0	1	1
1	0	1
1	1	1



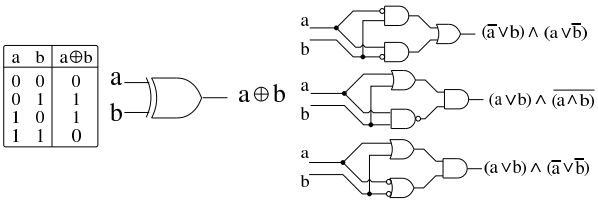
La porta logica AND, quando ha più di due ingressi, esprime l’equivalente di un’espressione in cui tutte le variabili in ingresso sono collegate dall’operatore OR:



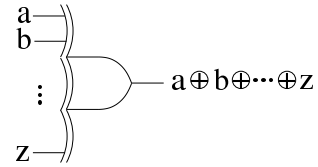
Connettivo XOR

Il connettivo XOR restituisce il valore *Vero* se solo uno dei due ingressi dispone di un valore pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero legge», l’espressione «A XOR B» equivale alla proposizione «Antonio mangia oppure Piero legge». Va comunque osservato che il connettivo XOR si considera derivato dagli altri e può essere tradotto in forme diverse, ma equivalenti.

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0



Anche la porta logica XOR può avere più ingressi, come avviene per AND e OR:



Ordine di precedenza

Le espressioni logiche vanno risolte tenendo conto che gli operatori hanno un ordine di precedenza: NOT, AND, OR. Per esempio, ***a***+***b***·***c’*** va inteso come ***a***+(***b***·(***c’***)). Naturalmente, l’ordine di precedenza può essere modificato utilizzando opportunamente le parentesi.

Valori irrilevanti nelle tabelle di verità

A volte si possono semplificare le tabelle di verità di una funzione logica quanto alcune variabili in ingresso, in certe circostanze, possono assumere qualsiasi valore senza interferire nel risultato; in quei casi, al posto del valore si mette convenzionalmente una lettera «x». A titolo di esempio, si consideri la funzione logica che si ottiene dall'espressione $A \cdot (B+C)$. Nella figura successiva si vede la tabella di verità completa e a fianco una versione semplificata della stessa.

Figura u97.11. Semplificazione delle tabelle di verità in presenza di condizioni in cui i valori di certe variabili diventano irrilevanti.

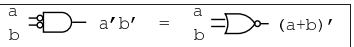
A	B	C	A(B+C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

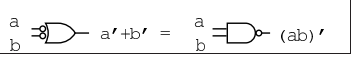
=

A	B	C	A(B+C)
0	**x**	**x**	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Equivalenze

Attraverso gli operatori logici si possono costruire delle espressioni complesse, nelle quali esiste la facoltà di applicare delle trasformazioni, di solito con lo scopo di cercare di semplificarle, mantenendo però lo stesso risultato. Per questo sono molto importanti i teoremi di De Morgan, secondo i quali:

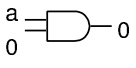
$a' \cdot b' = (a+b)'$ 

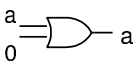
$a'+b' = (a \cdot b)'$ 

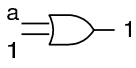
Mettendo assieme le tabelle della verità dei vari operatori logici con i teoremi di De Morgan, si ottengono le equivalenze della tabella successiva su cui si basa l'algebra «logica», ovvero l'algebra di Boole.

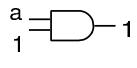
Tabella u97.13. Equivalenze dell'algebra di Boole. Si intende che ab rappresenti semplicemente $a \cdot b$ in modo compatto.

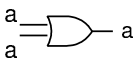
(1)	$a+0 = a$	(2)	$a \cdot 0 = 0$
(3)	$a+1 = 1$	(4)	$a \cdot 1 = a$
(5)	$a+a = a$	(6)	$a \cdot a = a$
(7)	$a+a' = 1$	(8)	$a \cdot a' = 0$
(9)	$a+b = b+a$	(10)	$ab = ba$
(11)	$a+(b+c) = (a+b)+c$	(12)	$a(bc) = (ab)c$
(13)	$a(b+c) = ab+ac$	(14)	$a+bc = (a+b)(a+c)$
(15)	$(a+b)' = a'b'$	(16)	$(ab)' = a'+b'$
(17)	$(a')' = a$		

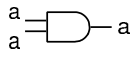
(1) a·0=0 

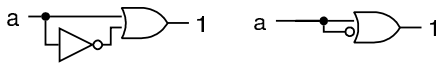
(2) a+0=a 

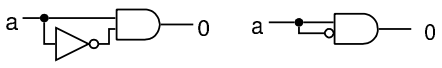
(3) a+1=1 

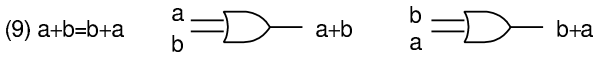
(4) a·1=1 

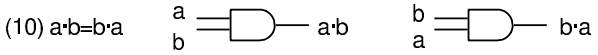
(5) a+a=a 

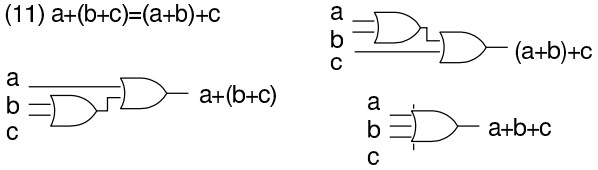
(6) a·a=a 

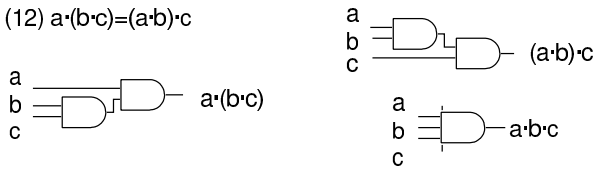
(7) a+a'=1 

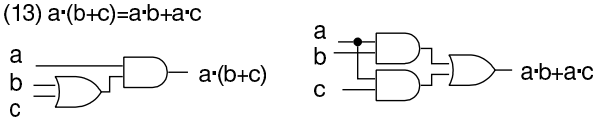
(8) a·a'=0 

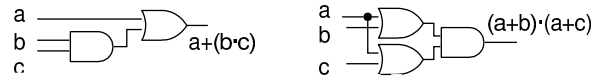
(9) a+b=b+a 

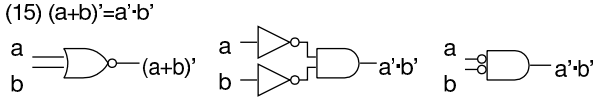
(10) a·b=b·a 

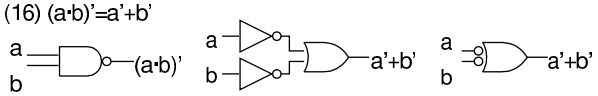
(11) a+(b+c)=(a+b)+c 

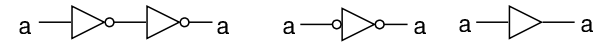
(12) a·(b·c)=(a·b)·c 

(13) a·(b+c)=a·b+a·c 

(14) a+(b·c)=(a+b)·(a+c) 

(15) (a+b)'=a'·b' 

(16) (a·b)'=a'+b' 

(17) (a')'=a 

Quando nelle espressioni si utilizzano solo gli operatori logici tradizionali (AND, OR, NOT), i teoremi di De Morgan consentono di trasformare facilmente una funzione logica nel suo complemento. In pratica, data la funzione f, si ottiene la negazione logica f' seguendo la procedura seguente:

- si scambiano gli operatori AND originali con gli operatori OR, mettendo delle parentesi per non modificare l'ordine di valutazione;
- si scambiano gli operatori OR originali con gli operatori AND, facendo in modo di evitare che questo cambiamento cambi l'ordine di esecuzione della valutazione;
- si complementano tutte le variabili (operatore NOT).

Si veda l'esempio seguente:

$f = a+b'c+de'+g'h'$

$f' = a'(b+c')(d'+e)(g+h)$

Somma dei prodotti

Una funzione logica è un'espressione composta da variabili, costanti logiche e operatori logici, la quale produce un risultato logico (*Vero* o *Falso*). Tuttavia, una funzione logica può essere descritta semplicemente attraverso la tabella di verità che abbina i valori delle variabili al risultato atteso per ogni combinazione delle stesse.

È possibile sintetizzare il comportamento di una funzione logica attraverso ciò che si definisce come ***somma dei prodotti***: si parte dalla tabella di verità che si vuole ottenere e si elencano i ***prodotti fondamentali*** logici che descrivono ogni condizione degli ingressi.

Tabella u97.14. Prodotti fondamentali delle funzioni aventi due, tre e quattro variabili.

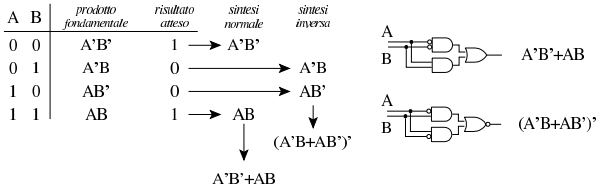
A	B	*prodotto fondamentale*
0	0	A'B'
0	1	A'B
1	0	AB'
1	1	AB

A	B	C	*prodotto fondamentale*
0	0	0	A'B'C'
0	0	1	A'B'C
0	1	0	A'BC'
0	1	1	A'BC
1	0	0	AB'C'
1	0	1	AB'C
1	1	0	ABC'
1	1	1	ABC

A	B	C	D	*prodotto fondamentale*
0	0	0	0	A'B'C'D'
0	0	0	1	A'B'C'D
0	0	1	0	A'B'CD'
0	0	1	1	A'B'CD
0	1	0	0	A'BC'D'
0	1	0	1	A'BC'D
0	1	1	0	A'BCD'
0	1	1	1	A'BCD
1	0	0	0	AB'C'D'
1	0	0	1	AB'C'D
1	0	1	0	AB'CD'
1	0	1	1	AB'CD
1	1	0	0	ABC'D'
1	1	0	1	ABC'D
1	1	1	0	ABCD'
1	1	1	1	ABCD

Per esempio, la funzione nota come NXOR (funzione di due variabili che si avvera solo quando le due variabili hanno lo stesso valore) si sintetizza secondo il procedimento che si può comprendere intuitivamente dalla figura successiva.

Figura u97.15. Sintesi a partire dalla tabella di verità della funzione NXOR, attraverso gli operatori logici fondamentali.



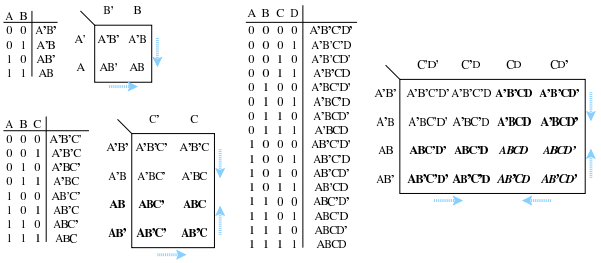
L'esempio della figura permette di ottenere due alternative, secondo quello che è noto come «somma dei prodotti», ma ciò non toglie che ci possano essere altre possibilità di sintesi.

Mappe di Karnaugh

Una funzione logica può essere rappresentato in una mappa di Karnaugh, a partire dalla sua tabella di verità; attraverso tale mappa si può poi cercare un modo per semplificare l'espressione che sintetizza la funzione e di conseguenza il circuito che la realizza.

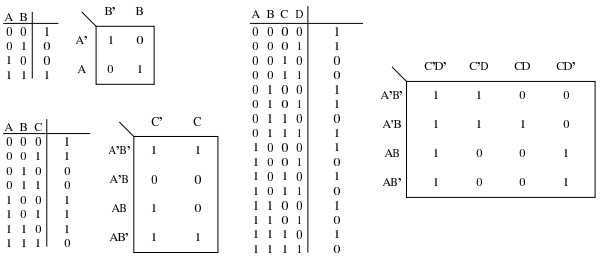
La mappa di Karnaugh è una rappresentazione bidimensionale dei prodotti fondamentali di una tabella di verità da analizzare: in pratica si dividono a metà i prodotti fondamentali e si rappresentano in un rettangolo. Nelle immagini successive si mostrano tre mappe, vuote, relative al caso di due, tre e quattro variabili: bisogna fare molta attenzione alla sequenza dei prodotti fondamentali, perché questa non corrisponde a quella che si usa normalmente nelle tabelle della verità.

Figura u97.16. Mappe di Karnaugh contenenti la definizione dei prodotti fondamentali a cui ogni cella fa riferimento.



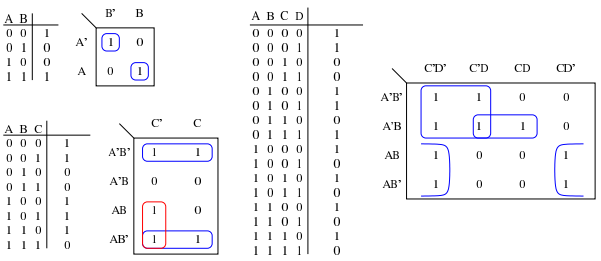
Le mappe vanno compilate mettendo nelle celle la cifra 1 in corrispondenza dei prodotti fondamentali validi. Le immagini successive propongono degli esempi, confrontando la tabella di verità con la mappa compilata corrispondente.

Figura u97.17. Mappe di Karnaugh compilate in base alle tabelle di verità che appaiono al loro fianco.



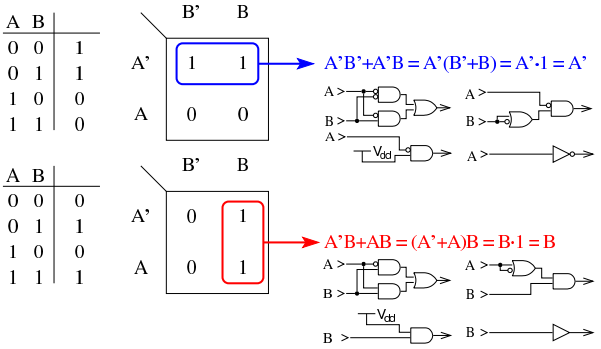
Una volta compilata la mappa, si vanno a raggruppare le celle che contengono la cifra 1 e che si trovano adiacenti in senso orizzontale o verticale.

Figura u97.18. Mappe di Karnaugh con le celle raggruppate verticalmente e orizzontalmente.



Quando due celle adiacenti, in verticale o in orizzontale, contengono il valore 1, una variabile che riguarda le due celle diventa inutile. La figura successiva dimostra intuitivamente il procedimento.

Figura u97.19. Semplificazione in presenza di due celle adiacenti con valore 1.



La semplificazione deriva dal fatto che ***x*** OR ***x'*** è sempre vero; pertanto, quando un raggruppamento rettangolare dimostra che una variabile appare sia nella sua forma normale, sia negata, significa che si può semplicemente ignorare.

Figura u97.20. Esempi di semplificazioni con mappe a due e tre variabili.

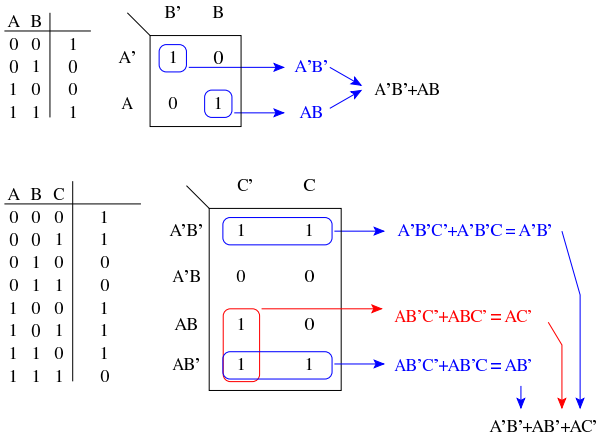
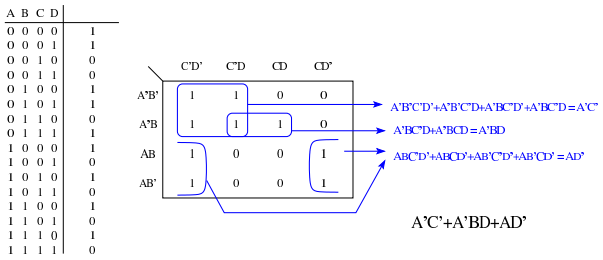
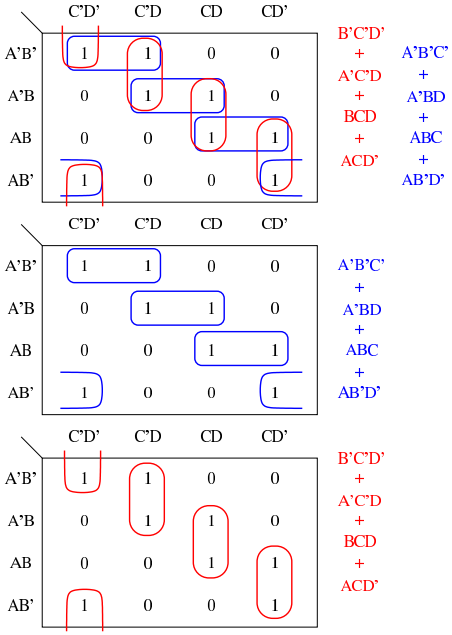


Figura u97.21. Esempio di semplificazione con mappa a quattro variabili.



Nella delimitazione delle zone che consentono di ottenere una semplificazione logica, si possono ignorare le zone che si sovrappongono completamente con altre, come si vede nella figura successiva, dove si possono produrre due risultati alternativi equivalenti.

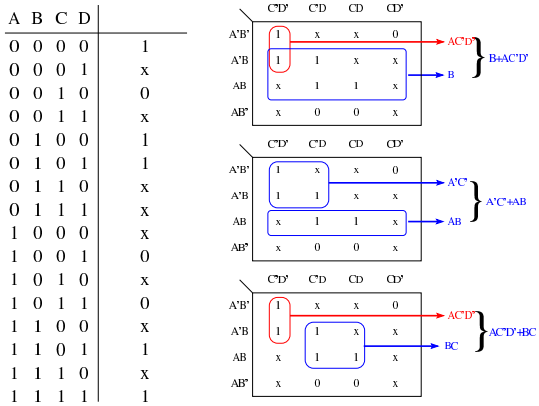
Figura u97.22. Esempi di semplificazioni in presenza di aree sovrapposte e ridondanti.



Le mappe di Karnaugh permettono di semplificare facilmente una funzione logica, purché non si superino le quattro variabili: diversamente sarebbe necessaria una rappresentazione a tre o più dimensioni.

In situazioni particolari, può accadere che sia indifferente il valore prodotto da alcune combinazioni delle variabili di ingresso. In questi casi, per sintetizzare la rete combinatoria si può scegliere liberamente il valore di uscita che risulta più conveniente nell'ottica della semplificazione.

Figura u97.23. Esempio di semplificazione in presenza di esiti indifferenti; si mostrano tre ipotesi di soluzione.



Le mappe di Karnaugh si possono usare anche in presenza di più funzioni che condividono le stesse variabili di ingresso. In tal caso si valutano tante mappe quante sono le funzioni per cercare una sintesi per ognuna; successivamente, se si realizzano queste funzioni attraverso un circuito logico, si cercano elementi comuni per condividerli senza duplicarli.

Figura u97.24. Due funzioni delle stesse variabili, sintetizzate attraverso due mappe.

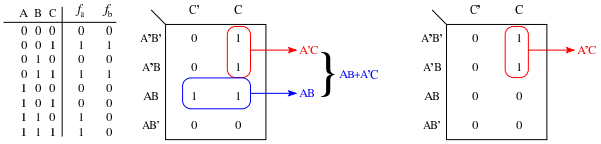
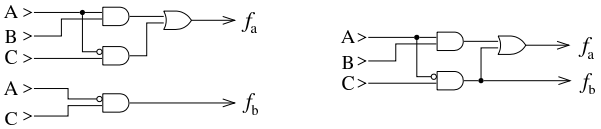


Figura u97.25. Soluzione in forma di circuito logico, prima separata, poi unita risparmiando una porta logica.



Mappe di Karnaugh con la funzione XOR

Le mappe di Karnaugh danno soluzioni formate attraverso gli operatori logici fondamentali (AND, OR, NOT); tuttavia, spesso è comodo avvalersi dell'operatore XOR, al pari di quelli fondamentali. Nelle mappe di Karnaugh l'operatore XOR si manifesta in modo particolare, come si vede nella figura successiva.

Figura u97.26. Mappe di Karnaugh relative a funzioni XOR a due, tre e quattro variabili.

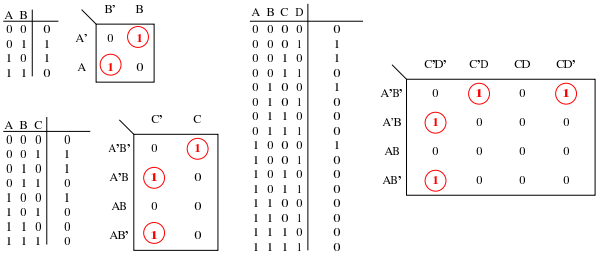


Figura u97.27. Esempi di individuazione di funzioni XOR a due variabili nelle mappe di Karnaugh a ingressi variabili. Le zone colorate di giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.

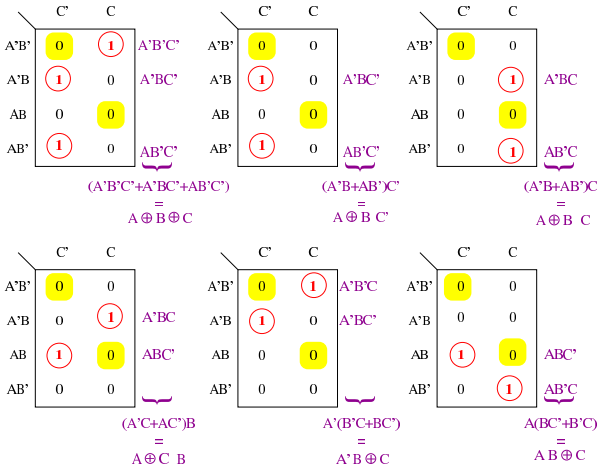


Figura u97.28. Esempi di individuazione di funzioni XOR a due variabili nelle mappe di Karnaugh a quattro ingressi: le zone in giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.

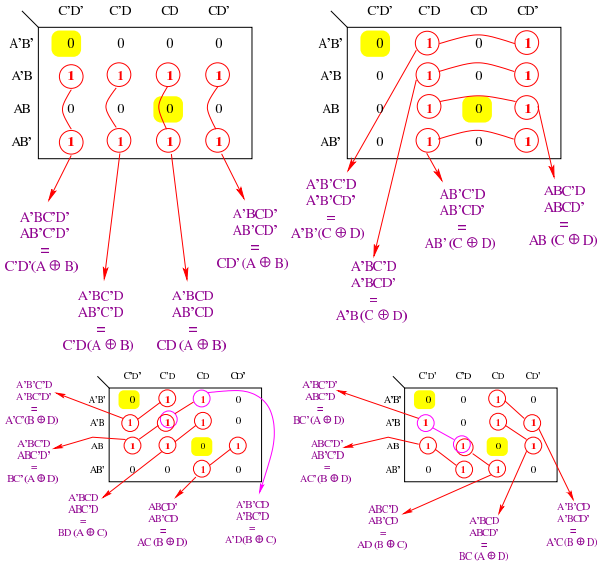
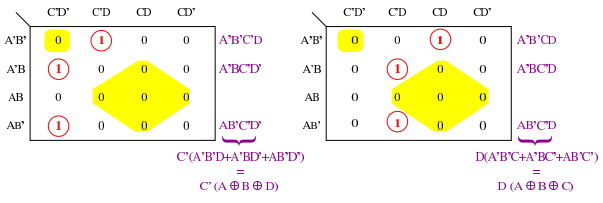
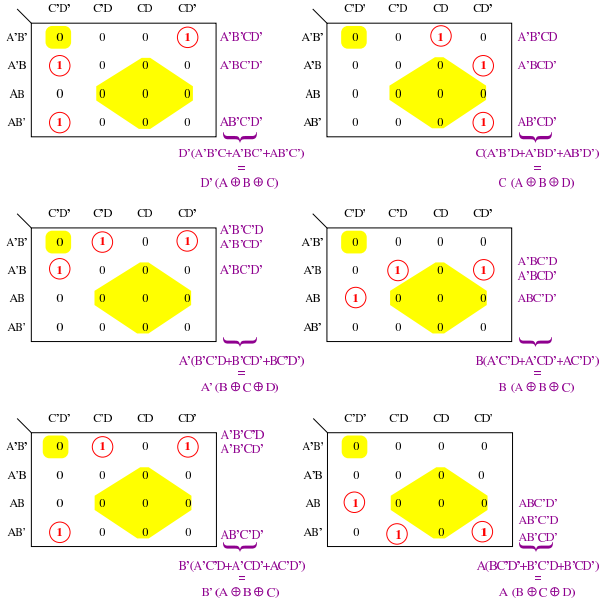


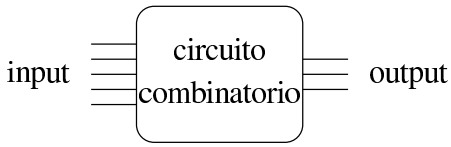
Figura u97.29. Esempi di individuazione di funzioni XOR a tre variabili nelle mappe di Karnaugh a quattro ingressi: le zone in giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.





Decodificatore 742
Demultiplatore 743
Multiplatore 744
Demultiplatori e multiplatori in parallelo 745
Codificatore binario 745
Codificatore di priorità 746
Unità logiche 747
Scorrimento 748
Addizionatore 750
Sottrazione 752
Somma e sottrazione assieme 753
Riporto anticipato 754
Complemento a due 756
Moltiplicazione 757
Divisione 761
Comparazione 763

Un ***circuito combinatorio***, ovvero una ***rete combinatoria***, è un sistema di porte logiche, connesse opportunamente tra loro, organizzato con un insieme di ingressi e un insieme di uscite, nel quale i valori logici delle uscite sono determinati direttamente e univocamente dai valori logici presenti negli ingressi. Il circuito combinatorio si può rappresentare, complessivamente, come una scatola composta da ingressi e da uscite, con una tabella di verità che stabilisce i valori delle uscite in base ai valori degli ingressi.

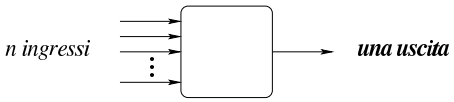


Quello raffigurato sopra è un esempio di circuito combinatorio con cinque ingressi e tre uscite, ma la proporzione tra quantità di ingressi e quantità di uscite dipende solo dalla funzione che deve realizzare tale circuito, ovvero dallo scopo che con questo ci si prefigge di raggiungere.

Si osservi che un circuito combinatorio, per essere tale, non deve essere influenzato dalla variabile tempo e nemmeno dalla variabile casuale dovuta all'accensione del circuito; pertanto, su tali circuiti non si considera il problema del tempo di propagazione, necessario a far sì che le uscite raggiungano i valori previsti in base ai valori pervenuti in ingresso.

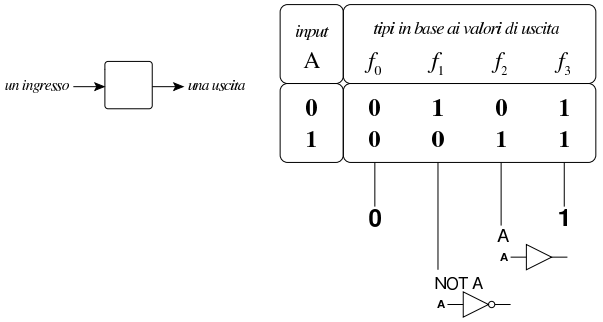
I circuiti combinatori più semplici sono quelli che dispongono di una sola uscita e realizzano quindi delle porte logiche, più o meno complesse.

Figura u98.2. Circuiti combinatori con una sola uscita.



Per comprendere la questione, si può osservare che un circuito composto da un solo ingresso e da una sola uscita può essere di quattro tipi differenti, come evidenziato dallo schema successivo.

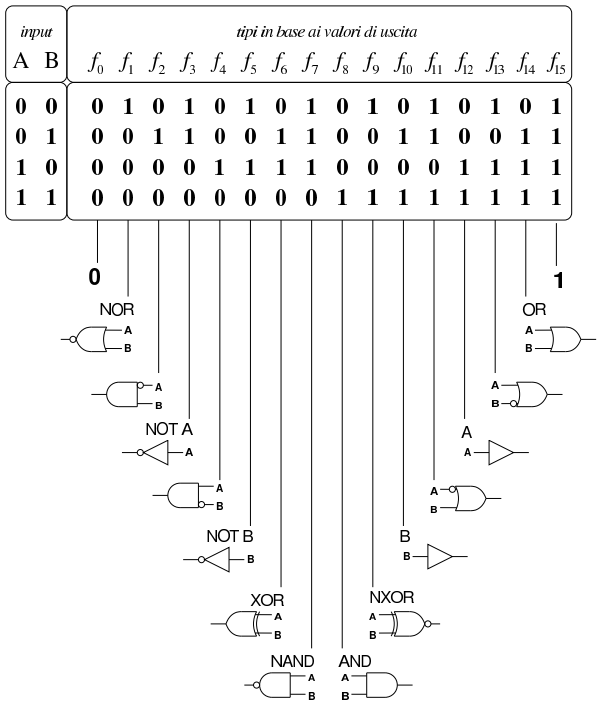
Schema u98.3. Tabella di verità per quattro funzioni di una variabile.



Come si vede dalle annotazioni contenute nello schema, il circuito corrispondente alla funzione f_1, coincide con il circuito invertente (ovvero NOT), mentre quello corrispondente alla funzione f_2 coincide con il circuito non-invertente.

In un circuito combinatorio con **due ingressi** e un'uscita, ci sono 16 funzioni possibili, come si vede nello schema successivo, dove si evidenziano in particolare le corrispondenze con le porte logiche comuni, indicate assieme al loro nome standard.

Schema u98.4. Tabella di verità per sedici funzioni di due variabili.



Decodificatore

Il decodificatore (*decoder*) è un circuito combinatorio che attiva una sola uscita, selezionandola in base alla combinazione di valori presenti negli ingressi, tenendo conto per ogni combinazione delle variabili di ingresso deve esserci un'uscita differente da attivare. Pertanto, per n ingressi ci sono 2^n uscite. Inoltre, solitamente, tale circuito combinatore è provvisto anche di un ingresso di controllo ulteriore, disattivando il quale si fa in modo che nessuna uscita risulti attiva, indipendentemente dagli altri valori in ingresso.

Figura u98.5. Schema a blocchi di un decodificatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso di abilitazione è indicato dalla sigla ***en*** (*enable*); le uscite sono indicate con le sigle da y_0 a y_3. Lo schema di destra è reso in una forma più compatta, dove gli ingressi di selezione risultano raccolti assieme.

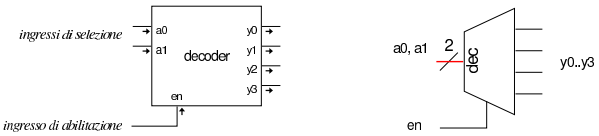
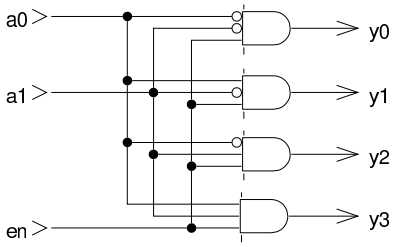


Tabella u98.6. Tabella di verità per un decodificatore a quattro uscite, da y_0 a y_3; due ingressi di selezione, a_0 e a_1; un ingresso di abilitazione ***en***.

en	a_1	a_0	y_3	y_2	y_1	y_0
0	***x***	***x***	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

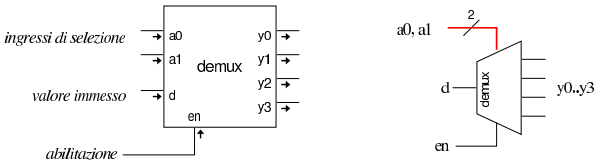
Figura u98.7. Esempio di realizzazione di un decodificatore a quattro uscite: i due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso di abilitazione è indicato dalla sigla ***en***; le uscite sono indicate con le sigle da y_0 a y_3.



Demultiplatore

Il ***demultiplatore*** (*demultiplexer*), spesso abbreviato con la sigla ***demux***, svolge un lavoro simile al decodificatore, ma in qualità di commutatore del valore contenuto in un ingresso aggiuntivo.

Figura u98.8. Schema a blocchi di un demultiplatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso dati corrisponde alla variabile d; l'ingresso di abilitazione è indicato dalla sigla ***en***; le uscite sono indicate con le sigle da y_0 a y_3. Nel disegno di destra si vede una versione più compatta.

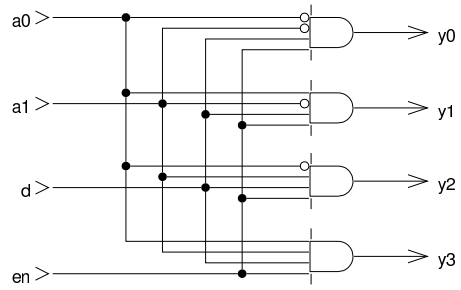


In questo caso, a differenza del decodificatore, l'uscita y_n selezionata riporta lo stesso valore dell'ingresso d.

Tabella u98.9. Tabella di verità per un demultiplatore a quattro uscite, da y_0 a y_3; due ingressi di selezione, a_0 e a_1; un ingresso dati d; un ingresso di abilitazione ***en***.

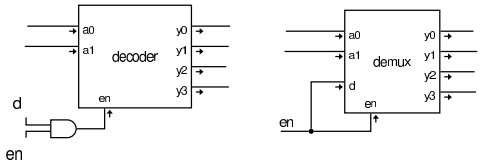
en	a_1	a_0	d	y_3	y_2	y_1	y_0
0	***x***	***x***	***d***	0	0	0	0
1	0	0	***d***	0	0	0	***d***
1	0	1	***d***	0	0	***d***	0
1	1	0	***d***	0	***d***	0	0
1	1	1	***d***	***d***	0	0	0

Figura u98.10. Esempio di realizzazione di un demultiplatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; il dato da commutare nell'uscita selezionata viene letto dall'ingresso ***d***; l'ingresso di abilitazione è indicato dalla sigla ***en***; le uscite sono indicate con le sigle da y_0 a y_3.



Il demultiplatore può essere ottenuto facilmente da un decodificatore munito di ingresso di abilitazione; in modo analogo, un demultiplatore può essere ridotto a funzionare come un decodificatore.

Figura u98.11. A sinistra un decodificatore adattato per funzionare come demultiplatore; a destra un demultiplatore adattato per funzionare come decodificatore.



Multiplatore

Il ***multiplatore*** (*multiplexer*), spesso abbreviato con ***mux***, è un circuito combinatorio che seleziona il valore di una sola porta di entrata e lo riproduce nell'uscita. La selezione della porta di entrata dipende dalla combinazione di valori contenuta in un insieme di porte di selezione. Per ***n*** porte di selezione si può scegliere tra 2^n porte di entrata.

Figura u98.12. Multiplatore a quattro entrate. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; le entrate hanno le sigle da d_0 a d_3; l'ingresso di abilitazione è indicato dalla sigla ***en*** (*enable*); l'uscita è indicate con la variabile ***y***. Lo schema di destra rappresenta una forma compatta, nella quale le variabili di selezione sono raccolte assieme in una linea multipla.

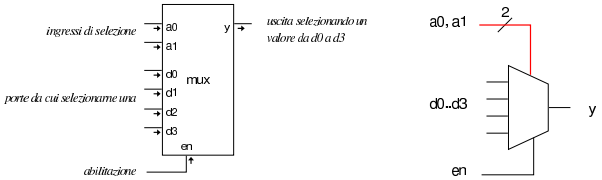
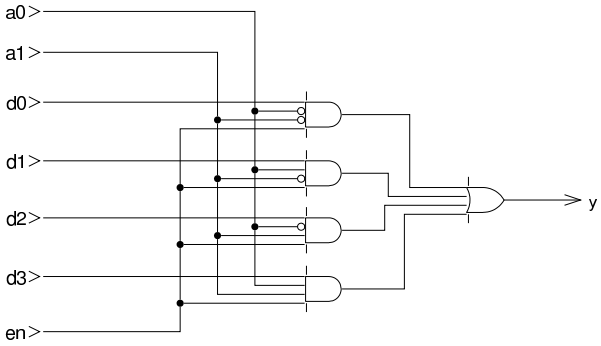


Tabella u98.13. Tabella di verità per un multiplatore a quattro ingressi di dati, da d_0 a d_3; due ingressi di selezione, da a_0 e a_1; un ingresso di abilitazione ***en***; un'uscita ***y***.

en	a_1	a_0	d_3	d_2	d_1	d_0	***y***
0	***x***	***x***	d_3	d_2	d_1	d_0	0
1	0	0	d_3	d_2	d_1	d_0	d_0
1	0	1	d_3	d_2	d_1	d_0	d_1
1	1	0	d_3	d_2	d_1	d_0	d_2
1	1	1	d_3	d_2	d_1	d_0	d_3

Figura u98.14. Esempio di realizzazione di un multiplatore a quattro entrate.



Demultiplatori e multiplatori in parallelo

Quando i demultiplatori o i multiplatori vengono usati per commutare dati composti da più bit e quindi trasportati da più linee, servono tanti demultiplatori o multiplatori quanti sono tali bit, mettendo però assieme tutti gli ingressi di selezione e di abilitazione. Tuttavia, nel disegno di un circuito tali linee multiple si annotato in forma compatta. Le figure successive mostrano un demultiplatore e un multiplatore che trattano dati a quattro bit.

Figura u98.15. Demultiplatore con entrata e uscite composte da vettori di quattro valori.

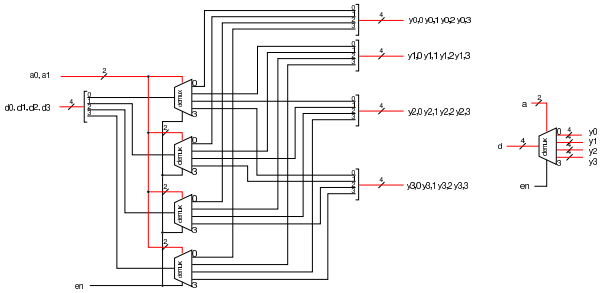
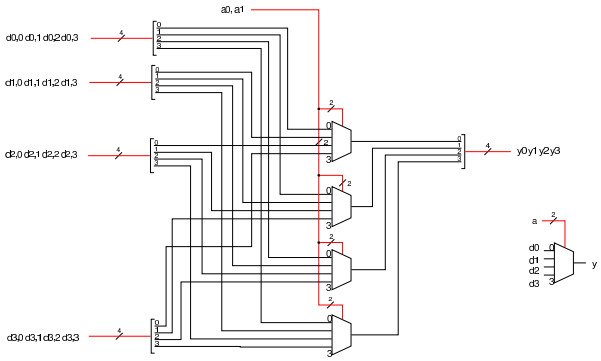


Figura u98.16. Multiplatore con entrate e uscita composte da vettori di quattro valori.



Codificatore binario

Un ***codificatore binario*** (*binary encoder* è un circuito combinatorio nei cui ingressi ci può essere una sola linea attiva e ha lo scopo di tradurre l'ingresso selezionato in un numero binario nelle uscite. In pratica, si tratta generalmente di 2^n ingressi e di ***n*** uscite. Vengono mostrati esempi di codificatori con ***n*** pari a 1, 2 e 3.

Tabella u98.17. Tabella di verità per un codificatore binario avente due ingressi e una uscita.

w_1	w_0	y_0
0	1	0
1	0	1

Figura u98.18. Il codificatore binario a due ingressi è un circuito inutile, in quanto basta collegare il secondo ingresso all'uscita per ottenere il risultato.

Tabella u98.19. Tabella di verità per un codificatore binario avente quattro ingressi e due uscite.

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Figura u98.20. Codificatore binario a quattro ingressi e due uscite. Il primo ingresso non viene collegato, perché in quella circostanza l'uscita è comunque pari a zero.

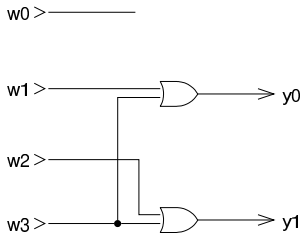
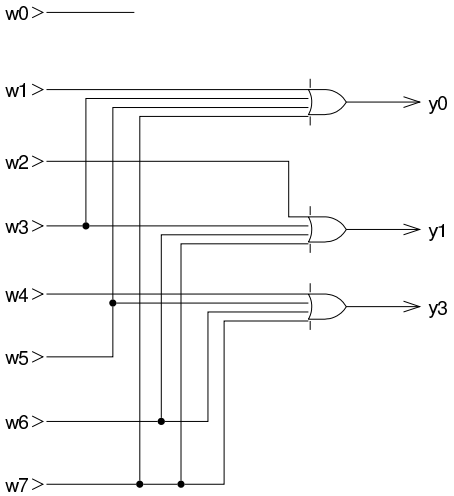


Tabella u98.21. Tabella di verità per un codificatore binario avente otto ingressi e tre uscite.

w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Figura u98.22. Codificatore binario a otto ingressi e tre uscite.



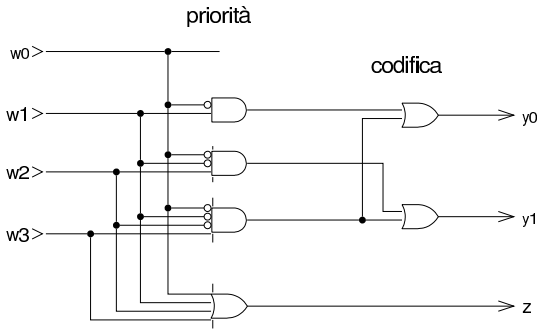
Codificatore di priorità

Il ***codificatore di priorità*** (*priority encoder*) è un codificatore binario che ammette l'attivazione di qualunque ingresso, emettendo il numero corrispondente all'ingresso attivo avente priorità rispetto agli altri. In questo caso è ammissibile anche il fatto che nessun ingresso sia attivo, pertanto, può essere presente un'uscita aggiuntiva che si attiva quando in ingresso c'è almeno un valore attivo.

Tabella u98.23. Tabella di verità per un codificatore di priorità avente quattro ingressi. L'ingresso attivo che ha indice più basso ha priorità rispetto agli altri, pertanto si usa la lettera ***x*** per indicare un valore indifferente della variabile corrispondente; l'uscita ***z*** si attiva se esiste almeno un ingresso attivo e quando l'uscita ***z*** è a zero, non ha importanza il valore delle altre uscite.

w_3	w_2	w_1	w_0	z	y_1	y_0
0	0	0	0	0	x	x
x	x	x	1	1	0	0
x	x	1	0	1	0	1
x	1	0	0	1	1	0
1	0	0	0	1	1	1

Figura u98.24. Codificatore di priorità a quattro ingressi.



Unità logiche

Con l'ausilio di un multiplatore è possibile ottenere facilmente un'unità logica, in grado di eseguire le operazioni logiche comuni, scegliendo quella desiderata attraverso un valore di selezione. Per esempio, si potrebbe realizzare un circuito che svolge il compito schematizzato in questo disegno:

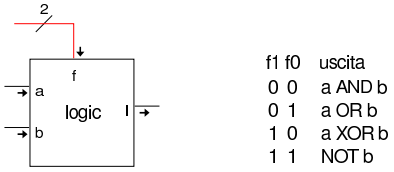
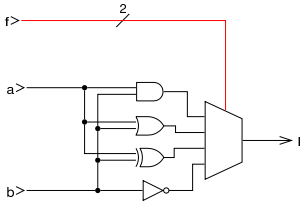
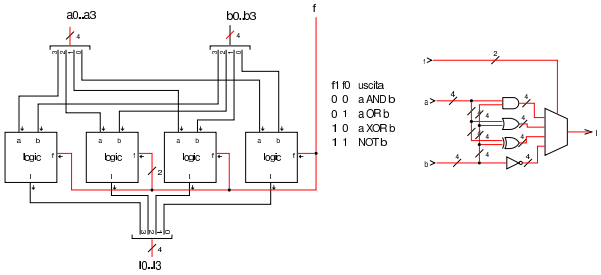


Figura u98.26. Esempio di realizzazione di un'unità logica con quattro opzioni.



Le unità logiche di questo tipo sono utili se si applicano a coppie di ingressi che dispongono, ciascuno, di più bit. Per ottenere questo risultato basta abbinare delle unità semplici, a una sola uscita.

Figura u98.27. Le unità logiche possono essere messe in parallelo per operare su interi binari a più cifre. Lo schema di destra è identico dal punto di vista circuitale, ma utilizza un modo compatto per rappresentare la duplicazione delle porte logiche in parallelo.



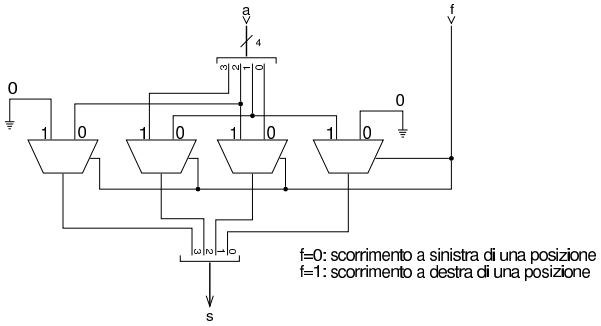
Scorrimento

Un tipo di operazione molto importante che si può realizzare con dei circuiti combinatori, è lo scorrimento dei bit. In pratica si ha un gruppo di n ingressi ordinati, da 0 a $n-1$, un gruppo di n uscite, ordinate nello stesso modo; eventualmente ci può essere un riporto in ingresso e uno in uscita. Spesso si considera lo spostamento di una sola unità, perché si possono ottenere spostamenti di più unità ripetendo la stessa operazione ciclicamente.

Il tipo più semplice di scorrimento è quello logico, nel quale lo spostamento a destra fa inserire uno zero nella posizione più significativa, mentre quello a sinistra lo fa entrare dalla parte destra. Nelle figure successive, per ottenere il valore zero si vede semplicemente un collegamento a massa, la quale si intende essere implicitamente al potenziale corrispondente allo zero.

1 1 1 0	*valore originale*	1 1 1 0	*valore originale*
1 1 0 0	*scorrimento logico a sinistra*	0 1 1 1	*scorrimento logico a destra*

Figura u98.29. Scorrimento logico, a sinistra o a destra, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso f. Lo scorrimento fa perdere una cifra e fa inserire uno zero dal lato opposto.

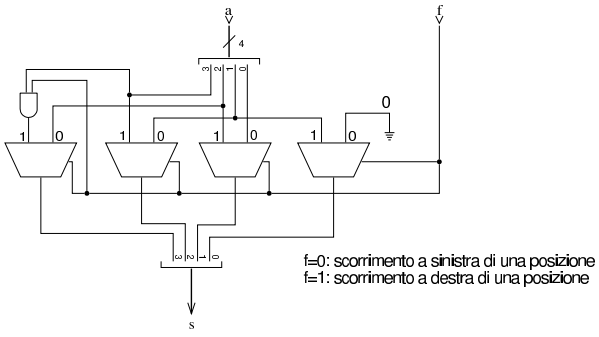


Lo scorrimento aritmetico avviene come lo scorrimento logico, con la differenza che lo scorrimento a destra deve mantenere inalterato il segno. In pratica, nello scorrimento a destra di tipo aritmetico, la cifra che viene immessa nella posizione più significativa, è la copia di quella che era originariamente in quella posizione. Lo scorrimento aritmetico si chiama così perché corrisponde alla moltiplicazione o alla divisione per due (la base di numerazione); va però osservato che lo scorrimento a sinistra può provocare un'inversione indesiderata di segno.

1 1 1 0	*valore originale*	1 1 1 0	*valore originale*
1 1 0 0	*scorrimento aritmetico a sinistra*	1 1 1 1	*scorrimento aritmetico a destra*
0 1 1 0	*valore originale*	0 1 1 0	*valore originale*
1 1 0 0	*scorrimento aritmetico a sinistra (*)*	0 0 1 1	*scorrimento aritmetico a destra*
1 0 1 0	*valore originale*	0 1 1 0	*valore originale*
0 1 0 0	*scorrimento aritmetico a sinistra (*)*	0 0 1 1	*scorrimento aritmetico a destra*

(*) nello scorrimento a sinistra si può produrre un'inversione di segno (overflow)

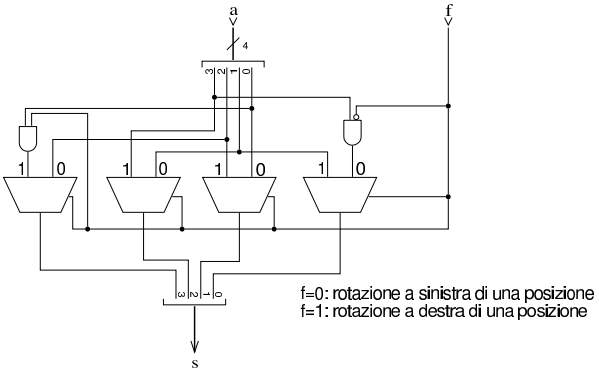
Figura u98.31. Scorrimento aritmetico, a sinistra o a destra, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso f.



Lo scorrimento circolare comporta il recupero della cifra che fuoriesce da una parte, dal lato opposto, sia nello scorrimento a sinistra, sia in quello a destra.

1 1 1 0	*valore originale*	1 1 1 0	*valore originale*
1 1 0 1	*rotazione a sinistra*	0 1 1 1	*rotazione a destra*
0 1 1 0	*valore originale*	0 1 1 0	*valore originale*
1 1 0 0	*rotazione a sinistra*	0 0 1 1	*rotazione a destra*
1 0 1 0	*valore originale*	1 0 1 0	*valore originale*
0 1 0 1	*rotazione a sinistra*	0 1 0 1	*rotazione a destra*

Figura u98.33. Scorrimento circolare di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso f.



Lo scorrimento circolare con riporto, utilizza il riporto preesistente per la cifra da immettere, da un lato o dall'altro, mentre mette nel riporto in uscita la cifra che viene espulsa.

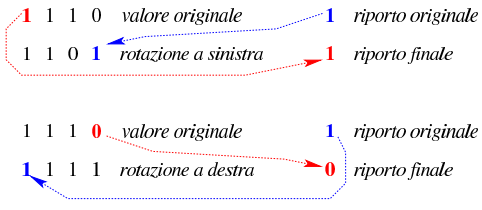
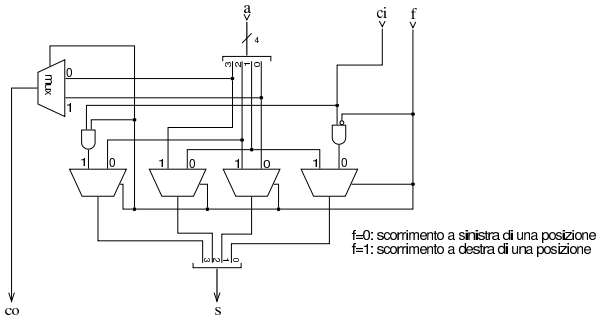


Figura u98.35. Scorrimento circolare con riporto, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso ***f***.



Addizionatore

«

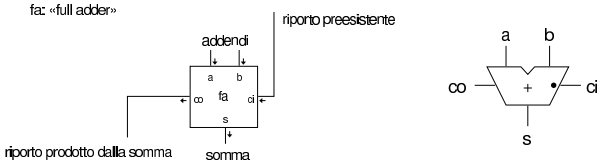
L'addizione in binario, eseguita con la stessa procedura consueta per il sistema di numerazione decimale, non genera mai un riporto superiore a uno. Lo si può verificare facilmente attraverso la tabella successiva.

Tabella u98.36. Addizione binaria.

primo addendo	secondo addendo	riporto preesistente	riporto generato	risultato
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Pertanto, il circuito combinatorio che può svolgere questo lavoro deve avere tre ingressi (primo addendo, secondo addendo, riporto preesistente) e due uscite (risultato e riporto generato). Di solito si usa la lettera ***c*** (*carry*) per indicare un riporto; in questo caso si distingue tra riporto in ingresso (***ci***) e riporto in uscita (***co***).

Figura u98.37. Schema a blocchi di un addizionatore. La sigla «fa» sta per *full adder*, ovvero «addizionatore completo».



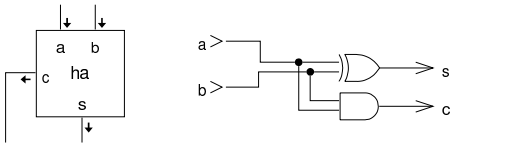
Si considera tradizionalmente che l'addizionatore completo (quello che si vede nello schema appena mostrato) sia costituito da due moduli più piccoli, noti come ***semiaddizionatori***, ovvero *half adder*. Il semiaddizionatore è un circuito combinatorio con due ingressi, ***a*** e ***b***, corrispondenti alle variabili da sommare, e due uscite, ***s*** e ***c***, corrispondenti al risultato della somma e al suo riporto.

Tabella u98.38. Tabella di verità per il semiaddizionatore.

a	***b***	***c***	***s***
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Si può intuire facilmente che, nel semiaddizionatore, l'uscita ***s*** si ottenga con una porta XOR e che il riporto si ottenga con una porta AND.

Figura u98.39. Schema a blocchi ed esempio di realizzazione di un semiaddizionatore. La sigla «ha» sta per *half adder*.



Per arrivare a ottenere l'addizionatore completo, occorre sommare anche il riporto precedente.

Figura u98.40. Schema a blocchi dell'addizionatore completo, ottenuto attraverso due semiaddizionatori e realizzazione conseguente.

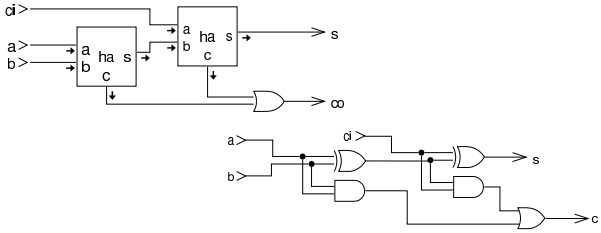
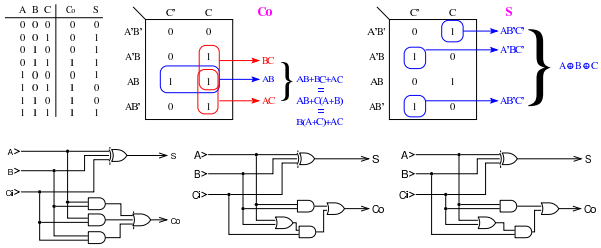
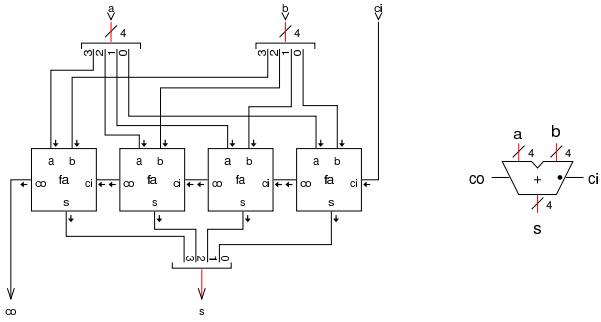


Figura u98.41. Soluzioni alternative per l'addizionatore completo, con l'aiuto delle mappe di Karnaugh.



Perché l'addizionatore sia utile, è necessario che intervenga su un numero binario composto da diverse cifre, pertanto occorre assemblare in parallelo più addizionatori, passando opportunamente il riporto, dalla cifra meno significativa a quella più significativa, una cifra alla volta.

Figura u98.42. Addizionatore a quattro cifre, costruito secondo la modalità della «propagazione del resto». Nella parte destra si vede una rappresentazione compatta di un addizionatore, senza specificare in che modo sia effettuato il calcolo.



Il circuito dell'addizionatore descritto, opera correttamente per la somma di numeri positivi o negativi, quando i numeri negativi si rappresentano attraverso la tecnica del complemento a due.

Il complemento a due che serve a trasformare il sottraendo, si ottiene con il complemento a uno del suo valore, sommandogli una unità attraverso il riporto in ingresso.

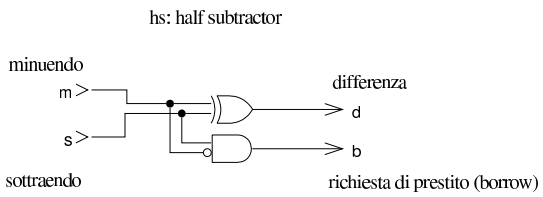
Sottrazione

Per la sottrazione, si parte, anche in questo caso, dal semi-sottrattore, ovvero *half subtractor*. Trattandosi di una sottrazione, è necessario distinguere tra gli operandi il minuendo e il sottraendo; in pratica, negli esempi vengono usate queste variabili: $d=m-s$.

Tabella u98.43. Tabella di verità per il semisottrattore.

m	s	b	d
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Figura u98.44. Semisottrattore.

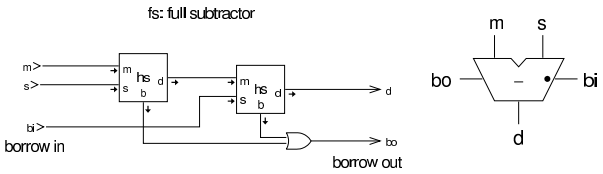


Al posto del riporto c'è un'uscita denominata *borrow* che rappresenta la richiesta del prestito di una cifra. Questa uscita diventa attiva quando il minuendo è pari a zero, mentre il sottraendo è pari a uno; ovvero quando il sottraendo è maggiore del minuendo. Il sottrattore completo ha un ingresso in più, costituito dalla richiesta di una cifra proveniente dallo stadio precedente; per gestire questo nuovo ingresso si possono usare due semisottrattori, dove il secondo sottrae al risultato del primo la richiesta di prestito.

Tabella u98.45. Tabella di verità per il semisottrattore.

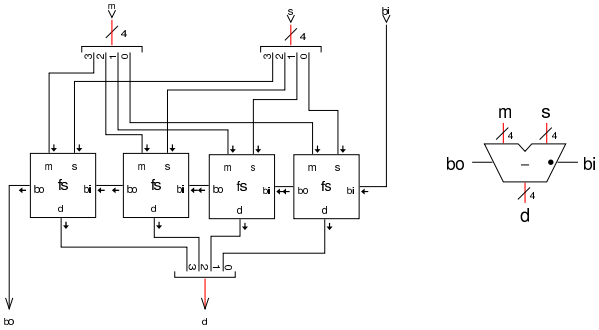
m	s	b_i	b_o	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figura u98.46. Sottrattore completo.



Come nel caso della somma, si possono creare delle catene di sottrattori completi; tuttavia, nel confronto con il circuito della somma, modificato per effettuare la sottrazione, la linea *borrow* (quella della richiesta del prestito), funziona in modo inverso, in quanto se attiva, rappresenta una cifra da togliere.

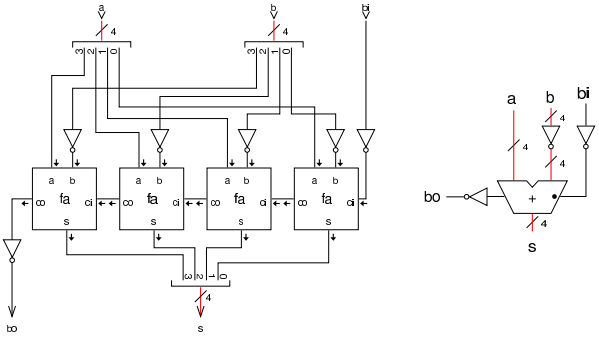
Figura u98.47. Sottrazione su più bit simultaneamente.



Somma e sottrazione assieme

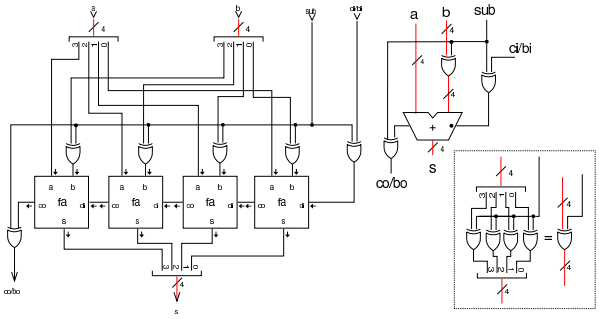
Il circuito dell'addizionatore opera correttamente per la somma di numeri positivi o negativi, quando i numeri negativi si rappresentano attraverso la tecnica del complemento a due; pertanto, per trasformare l'addizionatore in un circuito che invece sottrae uno dei due operandi, è sufficiente calcolare per quello il complemento a due e per ottenerlo si devono invertire tutti gli ingressi, compreso il riporto. Va ricordato che nella sottrazione, il riporto assume il significato della richiesta di una cifra.

Figura u98.48. Sottrattore a quattro cifre: in questo caso si tratta precisamente di $a-b$. Nella parte destra si vede una rappresentazione compatta della stessa cosa, dove il simbolo della porta NOT va intesa come un'abbreviazione di quattro porte NOT indipendenti.



Eventualmente, si trasforma facilmente il circuito combinatorio in un complesso unico, in grado di sommare o di sottrarre, sfruttando una porta XOR al posto della porta NOT, aggiungendo un ingresso ulteriore che permetta di stabilire se si esegue una somma o se l'operando stabilito va invece sottratto.

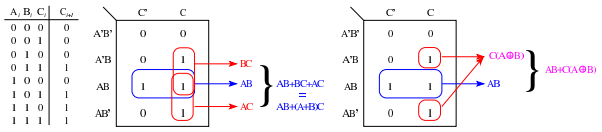
Figura u98.49. Somma o sottrazione a quattro cifre. Nella parte destra si vede una rappresentazione compatta della stessa cosa, dove il simbolo della porta XOR va inteso come un'abbreviazione di quattro porte XOR collegate assieme come si vede nel piccolo riquadro esplicativo.



Riporto anticipato

Nella somma di una quantità significativa di bit, la propagazione del riporto richiede un tempo relativamente elevato, dato che la somma di una certa cifra è corretta solo se è già avvenuta la somma di quelle che la precedono. Per poter accelerare l'esecuzione della somma, è necessario che per ogni cifra si possa sapere, nel tempo più breve possibile, qual è il riporto generato fino allo stadio precedente. Nelle espressioni successive, le variabili A_i, B_i e C_i, rappresentano i due addendi e il riporto in ingresso dello stadio i; pertanto, il riporto generato da questo stadio è rappresentato dalla variabile C_{i+1}.

Figura u98.50. Sintesi alternative del calcolo del riporto.



Nella figura si vede che il riporto si può sintetizzare in vari modi e in uno di questi appare anche l'uso dell'operatore XOR. Di quelle mostrate nella figura si scelgono due espressioni equivalenti:

$$C_{i+1} = A_iB_i+(A_i+B_i)C_i = A_iB_i+(A_i\oplus B_i)C_i$$

Da queste espressioni, si dichiarano due variabili nuove, G_i e P_i, le quali stanno rispettivamente per «generazione» e «propagazione», per cui si definisce che il riporto C_{i+1} è prodotto come funzione delle variabili G_i, P_i e C_i.

$$C_{i+1} = G_i+P_iC_i$$

È evidente che G_i equivale a A_iB_i, mentre P_i può essere considerata pari a A_i+B_i oppure $A_i\oplus B_i$, indifferentemente. Se il primo riporto generato (C_1) si può ottenere come $C_1=G_0+P_0C_0$, il secondo si ottiene come $C_2=G_1+P_1(G_0+C_0P_0)$, e di conseguenza si può proseguire per determinare i riporti successivi. Vengono mostrate le soluzioni per i primi quattro riporti:

$$C_1 = G_0+P_0C_0$$

$$C_2 = G_1+P_1C_1$$
$$C_2 = G_1+P_1(G_0+P_0C_0)$$
$$C_2 = G_1+P_1G_0+P_1P_0C_0$$

$$C_3 = G_2+P_2C_2$$
$$C_3 = G_2+P_2(G_1+P_1G_0+P_1P_0C_0)$$
$$C_3 = G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$$

$$C_4 = G_3+P_3C_3$$
$$C_4 = G_3+P_3(G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0)$$
$$C_4 = G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$$

Per semplificare le espressioni, si definiscono P_n e G_n nel modo seguente:

$$P_n = P_{n-1}P_{n-2}P_{n-1}\cdots P_1P_0$$

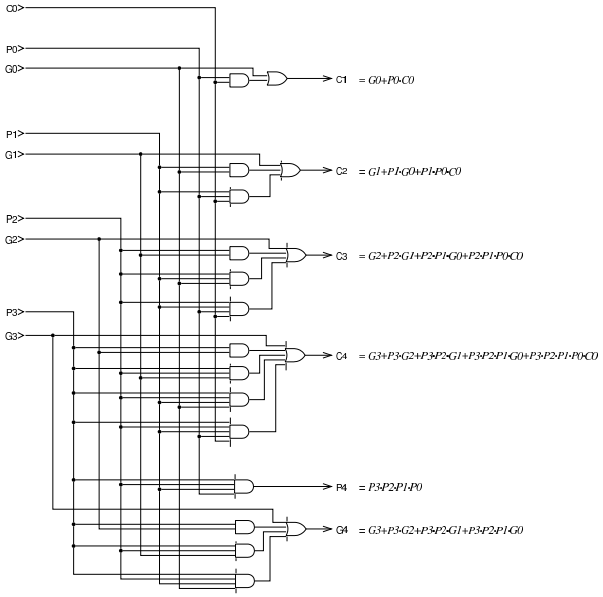
$$G_n = G_{n-1}+P_{n-1}G_{n-2}+P_{n-2}P_{n-1}G_{n-3}+\cdots+P_{n-2}P_{n-1}\cdots P_1G_0$$

Avendo definito ciò, il riporto C_n si può definire come:

$$C_n = G_n+P_nC_0$$

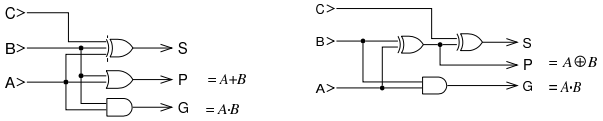
La figura successiva, mostra un circuito combinatorio che determina i riporti di quattro cifre binarie, partendo dal riporto iniziale e dai valori di $B_{3..0}$ e $G_{3..0}$, come descritto dalle equazioni che definiscono questa relazione. Il disegno contiene anche la logica necessaria a determinare il valore di B_4 e G_4 che possono servire per collegare assieme più moduli di questo tipo.

Figura u98.56. Schema per la determinazione di quattro riporti, a partire dal riporto iniziale (C_0) e dai valori di P_i e G_i, come descritto nelle equazioni logiche.



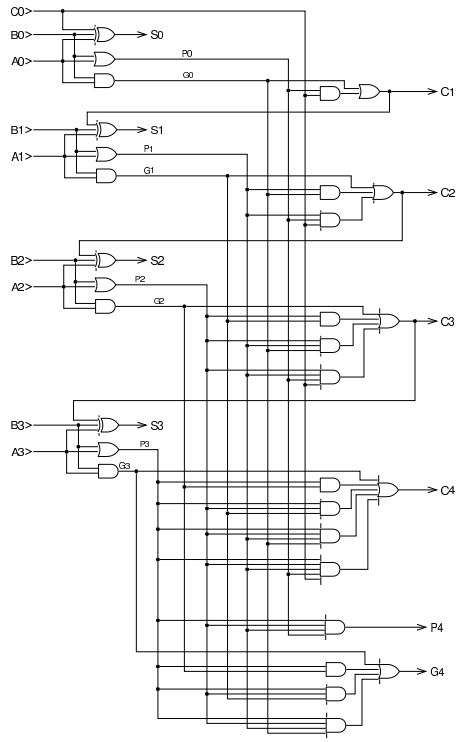
Avendo la necessità di disporre delle uscite G e P, si può sintetizzare un modulo per l'addizione privo della funzione che determina il riporto in uscita, dal momento che questo compito viene affidato al modulo che si vede nella figura precedente. Ci sono due soluzioni alternative, nelle quali il valore di P viene determinato nei due modi diversi già descritti con le tabelle di Karnaugh.

Figura u98.57. Moduli per l'addizione con le uscite P e G, ma privi dell'uscita con il riporto per la cifra successiva.



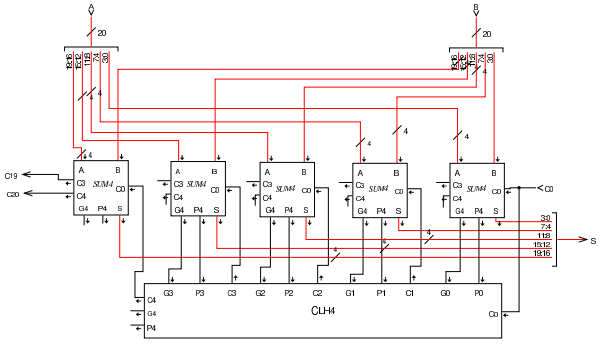
La figura successiva mostra un addizionatore a quattro cifre, dove si utilizzano i moduli di addizione e di determinazione del riporto già apparsi.

Figura u98.58. Addizionatore a quattro cifre.



È possibile collegare assieme i moduli mostrati, quando non si può disporre in partenza della quantità di cifre che servono. Questo collegamento comporta un aumento del ritardo di propagazione, ma si tratta comunque di un grande miglioramento rispetto al calcolo del riporto in cascata, come mostrato nelle sezioni precedenti. Nella figura successiva, il modulo SUM4 corrisponde allo schema di figura u98.58, dal quale si prelevano solo l'ultimo e il penultimo riporto (perché in sezioni successive viene mostrato che questi due consentono di determinare la presenza di uno straripamento); invece, il modulo CLH4 corrisponde allo schema di figura u98.56, il quale viene usato per mettere assieme i moduli di addizione.

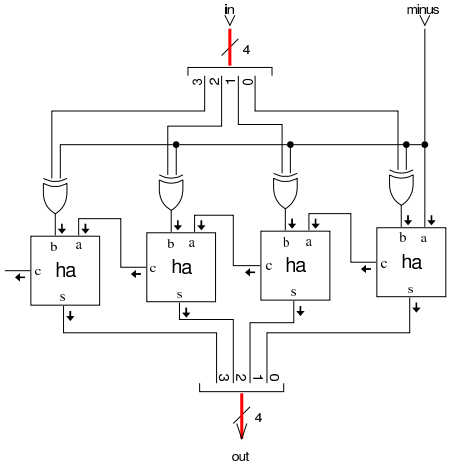
Figura u98.59. Addizionatore a 20 cifre, ottenuto partendo da moduli da quattro cifre concatenati assieme.



Complemento a due

Se si ha la necessità di invertire il segno di un numero intero, rappresentato in forma binaria, c'è bisogno di costruire un circuito che esegua il complemento a uno, invertendo le cifre binarie, sommando poi una unità per trovare il complemento a due.

Figura u98.60. Calcolo del complemento a due, corrispondente all'inversione del segno di un numero intero a quattro cifre.



L'esempio proposto nella figura mostra che il dato in ingresso, viene invertito (negato) se risulta attivo il segnale ***minus***, ma lo stesso segnale ***minus*** viene sommato, producendo alla fine il complemento a due. Se invece il segnale ***minus*** non fosse attivo, il dato in ingresso non verrebbe alterato e non ci sarebbe l'incremento di un'unità, per cui il risultato sarebbe lo stesso, senza variazione.

Moltiplicazione

La moltiplicazione binaria è un procedimento che richiede la somma e lo scorrimento. Per poter tenere conto del segno, il calcolo andrebbe fatto come se si operasse su una quantità doppia di cifre; per esempio, se moltiplicatore e moltiplicando sono di sole quattro cifre binarie, il risultato deve poter essere di otto cifre e il calcolo andrebbe fatto come se anche gli operandi fossero di questo rango. Si osservino gli esempi che appaiono nella figura successiva.

Figura u98.61. Moltiplicazione di numeri a quattro cifre binarie, senza segno e con segno.

moltiplicazione di interi senza segno	*moltiplicando negativo, moltiplicatore positivo*	*moltiplicando positivo, moltiplicatore negativo*	*moltiplicando negativo, moltiplicatore negativo*
00001011×	11111011×	00000111×	11111011×
00001101=	00000101=	11111101=	11111101=
00001011+	11111011+	00000111+	11111011+
000000000+	000000000+	000000000+	000000000+
001011000+	111011000+	000111000+	111011000+
010110000+	000000000+	001110000+	110110000+
000000000+	000000000+	011100000+	101100000+
000000000+	000000000+	111000000+	011000000+
000000000+	000000000+	110000000+	110000000+
000000000=	000000000=	100000000=	100000000=
10001111	11100111	11101011	00001111

Nella figura, l'esempio di sinistra mostra la moltiplicazione tra 11 e 13; trattandosi di numeri privi di segno, vanno aggiungi degli zeri nelle posizioni più significative e di conseguenza va svolta la moltiplicazione. Nel secondo esempio, i numeri vanno intesi con segno e si tratta della moltiplicazione tra -5 e +5; in questo caso, il numero che viene inteso come negativo, deve essere completato con cifre a uno, per mantenere il segno negativo, e di conseguenza la moltiplicazione ne tiene conto. Nel terzo esempio è il moltiplicatore a essere negativo: 7·-3. In tal caso è il moltiplicatore che viene completato con le cifre a uno nella parte più significativa, condizionando di conseguenza il calcolo della moltiplicazione. L'ultimo esempio è uguale al primo, con la differenza che i due valori sono intesi con segno, pertanto sono completati con cifre a uno. In conclusione, la moltiplicazione deve tenere conto del fatto che i numeri siano con segno o senza; nel caso lo siano, devono essere estesi nella porzione più significativa con la cifra necessaria a mantenere il segno che hanno.

Per risolvere il problema in forma di circuito combinatorio, occorre incrociare i valori di moltiplicando e moltiplicatore, verificando in ogni posizione utile la coincidenza di valori a uno. I due disegni successivi vanno sovrapposti idealmente, in quanto mostrano il concetto in due fasi: le uscite delle porte AND devono essere sommate verticalmente per generare il prodotto.

Figura u98.62. Intreccio tra moltiplicando e moltiplicatore, abbinato a delle porte AND.

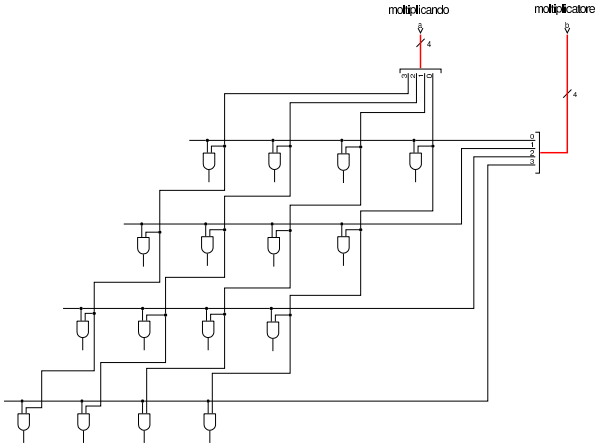
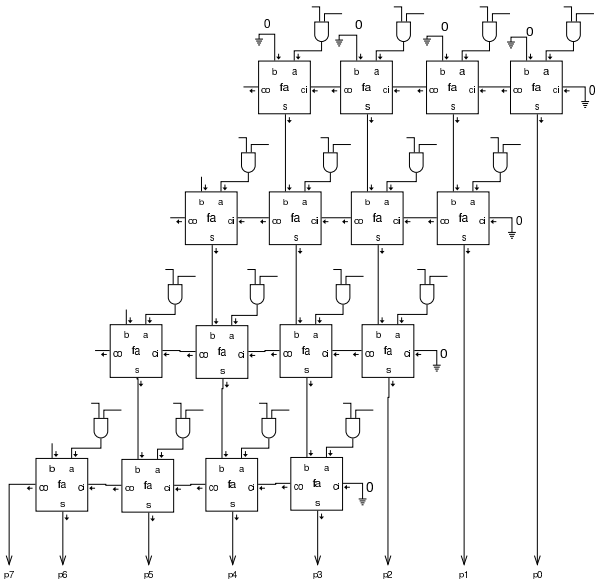
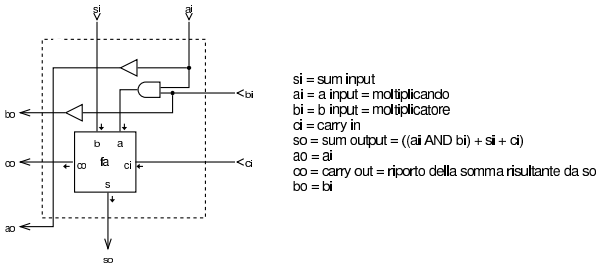


Figura u98.63. Quanto generato dalle porte AND, viene sommato con degli addizionatori completi.



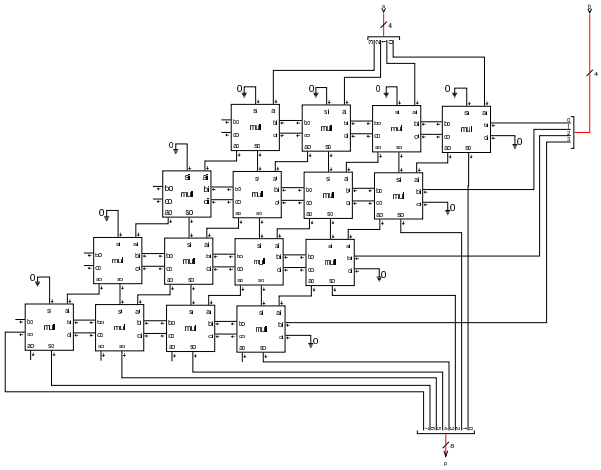
Per mettere assieme la moltiplicazione, svolta dall'operatore AND, e la somma, occorre costruire delle celle apposite, utilizzano un addizionatore completo. Come si vede dalla figura, uno degli addendi riceve il risultato del prodotto ottenuto con l'operatore AND, mentre l'altro addendo riporta il valore proveniente dalla riga superiore.

Figura u98.64. Cella usata per la costruzione della matrice che somma e fa scorrere il moltiplicando.



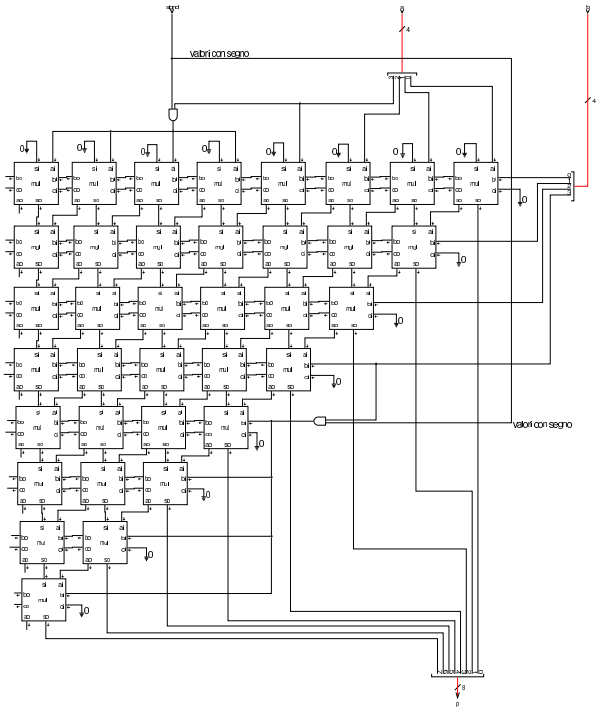
Le celle della figura vanno connesse per costruire le somme che costituiscono la moltiplicazione. La figura successiva mostra una soluzione limitata al caso di numeri privi di segno.

Figura u98.65. Moltiplicazione di interi **senza segno**, a quattro cifre, producendo un risultato a otto cifre: $\boldsymbol{p}=\boldsymbol{a}\cdot\boldsymbol{b}$.



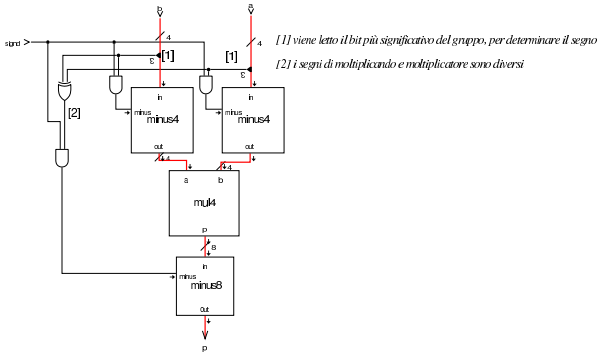
Per risolvere il problema dei valori con segno, occorre un ingresso aggiuntivo, per comunicare il fatto che si stia usando o meno numeri con segno, quindi occorrerebbe estendere la maglia come se si moltiplicassero valori con un numero doppio di cifre, estese secondo il segno che viene loro attribuito.

Figura u98.66. Soluzione completa, ma eccessivamente dispendiosa, per la moltiplicazione di due valori espressi a soli quattro bit.



Si può risolvere il problema della moltiplicazione quando si hanno valori con segno, adattare un circuito moltiplicatore di valori privi di segno, provvedendo a calcolare il complemento a due (ovvero a cambiare di segno) quando i valori risultano essere negativi. Nello schema della figura successiva, la funzione ***mul4*** corrisponde a un circuito moltiplicatore che opera solo su valori privi di segno; ***minus4*** e ***minus8*** sono circuiti che si occupano di invertire il segno se l'ingresso ***minus*** risulta attivo.

Figura u98.67. Adattamento di un moltiplicatore di valori senza segno, in modo da recepire anche valori con segno.



Nella figura si vede che dagli ingressi ***a*** e ***b***, viene prelevato il bit più significativo, per determinare se i valori rispettivi sono negativi; se lo sono, i loro valori vengono moltiplicati per -1, prima di passare alla moltiplicazione; al termine, il risultato viene moltiplicato per -1 solo se i segni di ***a*** e ***b*** sono diversi. Naturalmente, tutto questo è subordinato al fatto che venga richiesto un calcolo che tenga conto dei segni, attraverso l'ingresso ***signd***. Il circuito denominato ***minus4*** corrisponde a quanto mostrato nella sezione u0.13; naturalmente, ***minus8*** è solo un'estensione dello stesso a 8 bit.

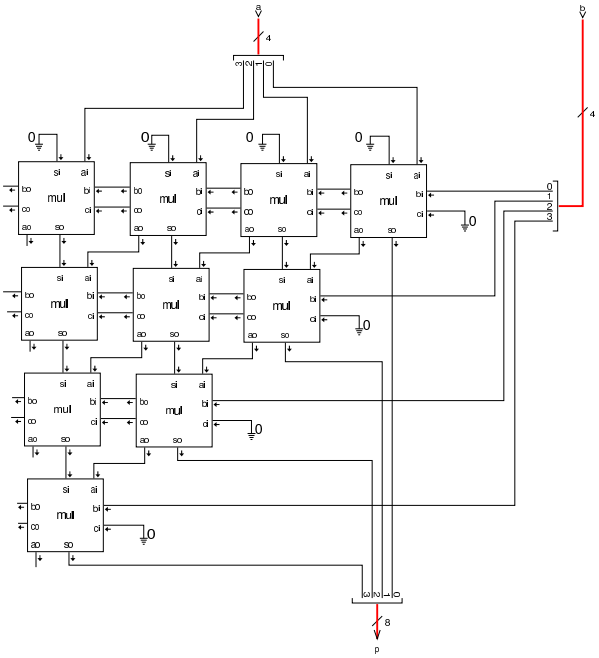
Se invece si riduce il risultato della moltiplicazione a una quantità di cifre uguale a quelle di moltiplicando e di moltiplicatore, ammesso che il risultato non venga troncato, ciò che si ottiene è valido,

indipendentemente dai segni, senza bisogno di dover comunicare la gestione dei segni al circuito. Si mostra nuovamente un esempio di calcolo della moltiplicazione, simile a quello iniziale, per dimostrare questa affermazione.

Figura u98.68. Moltiplicazione di numeri a quattro cifre binarie, troncando il risultato alla stessa ampiezza di moltiplicando e di moltiplicatore.

moltiplicazione di interi senza segno	*moltiplicando negativo, moltiplicatore positivo*	*moltiplicando positivo, moltiplicatore negativo*	*moltiplicando negativo, moltiplicatore negativo*
1011×	**1101**×	**0101**×	**1011**×
1101=	**0010**=	**1110**=	**1101**=
1011+	**0000**+	**0000**+	**1011**+
0000+	**1010**+	**1010**+	**0000**+
1100+	**00**00+	**01**00+	**11**00+
1000=	**0**000=	**1**000=	**1**000=
1111	1010	0110	1111
risultato non valido perché troncato troppo presto	*risultato valido*	*risultato valido*	*risultato valido*

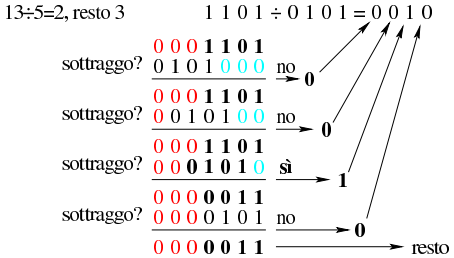
Figura u98.69. Circuito combinatorio semplificato per il prodotto tra due numeri, senza dover tenere conto della gestione del segno, ma con il risultato troncato al rango di moltiplicando e moltiplicatore.



Divisione

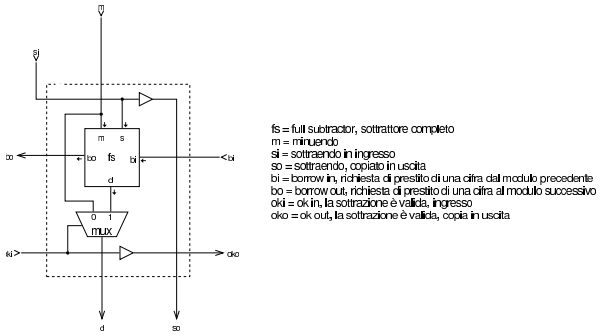
La divisione richiede la sottrazione e lo scorrimento, come si può vedere dall'esempio successivo, in cui il divisore viene confrontato con il dividendo, partendo dalle cifre più significative, sottraendo il divisore al dividendo solo quando ciò è possibile. In tal modo, il risultato intero della divisione è dato dal successo o meno di tali sottrazioni, mentre il resto è ciò che rimane dopo tutte le sottrazioni.

Figura u98.70. Procedimento usato per la divisione intera: dividendo e divisore si intendono privi di segno.



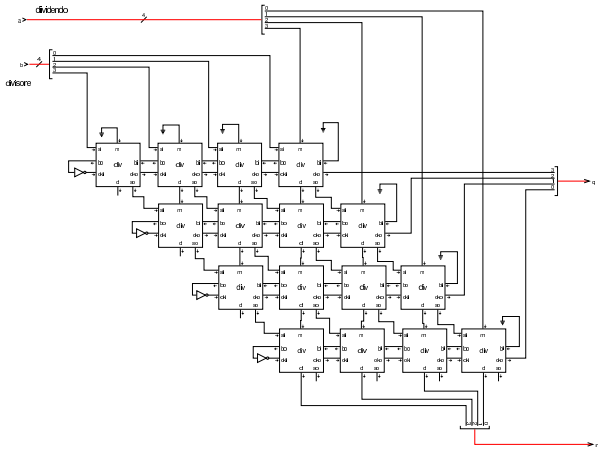
Per risolvere il problema attraverso un circuito combinatorio, si possono usare dei moduli per la sottrazione completa, da integrare con un controllo sul risultato, il quale deve essere prodotto solo se la sottrazione è ammissibile, altrimenti va riproposto il valore originale anche in uscita. Come nel caso della moltiplicazione, si possono costruire delle celle apposite.

Figura u98.71. Cella usata per la sottrazione condizionata.



Con queste celle, dopo ogni sottrazione, si ottiene l'informazione se c'è la richiesta di un prestito o meno, dall'uscita ***bo*** (*borrow out*: se c'è tale richiesta di prestito, significa che la sottrazione non può avere luogo, quindi, tale segnale viene invertito e usato per avvisare tutte le celle di una fila che devono riproporre il valore originale anche in uscita, rinunciando alla sottrazione.

Figura u98.72. Esempio di soluzione per la divisione di valori privi di segno.

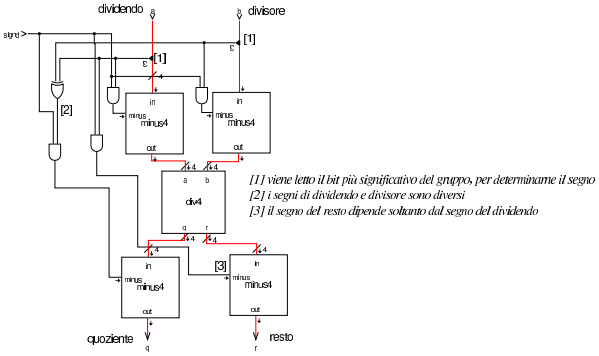


La divisione è complicata al riguardo della gestione dei valori con segno. Si pongono i casi seguenti, con cui stabilire il segno che devono avere i risultati:

Dividendo	Divisore	Quoziente	Resto
+	+	+	+
+	−	−	+
−	+	−	−
−	−	+	−

Per risolvere il problema della divisione di valori con segno, conviene aggiungere a ciò che divide valori senza segno, un controllo che inverta opportunamente i segni di dividendo, divisore, quoziente e resto, quando necessario.

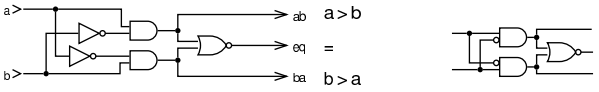
Figura u98.74. Completamento del divisore con la gestione dei valori con segno: ***div4*** è il divisore di valori privi di segno; ***minus4*** si occupa di invertire il segno del valore in ingresso, se il segnale ***signd*** è attivo.



Comparazione

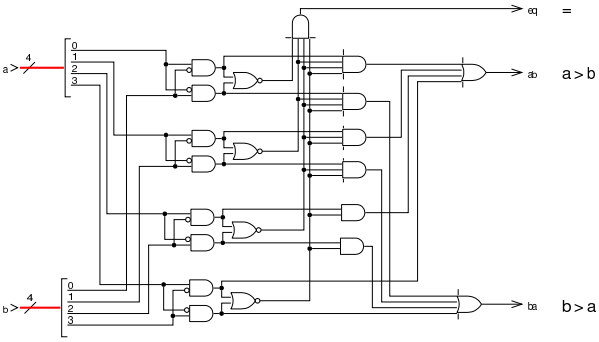
La comparazione tra due valori espressi in forma binaria, richiede il confronto, bit per bit, partendo dalla cifra più significativa. Da ogni confronto bit per bit, occorre sapere se sono uguali o se uno dei due sia maggiore. La figura successiva mostra la realizzazione di questo confronto tra due soli valori logici.

Figura u98.75. Confronto tra due soli ingressi; nel disegno di destra si usa una forma compatta per rappresentare gli ingressi negati delle due porte AND.



Nel confronto tra più bit, si fanno le stesse verifiche a coppia, facendo prevalere la prima differenza a partire dal bit più significativo.

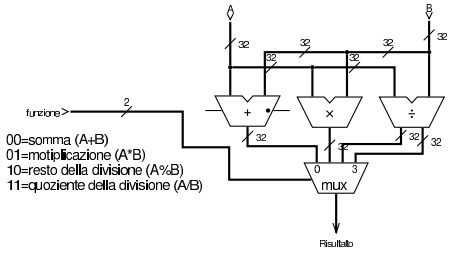
Figura u98.76. Confronto tra numeri senza segno, a quattro bit.



Indicatori 765
Troncamento di un valore in base al rango 766
Inversione di segno 767
Somma e sottrazione 768
Scorrimento e rotazione 768
Comparazione di valori 770
Moltiplicazione 772
Divisione 774
Unità logica 776
ALU completa 777

L'unità aritmetico-logica, nota come ALU (*arithmetic-logic unit*), è un circuito combinatorio che racchiude le funzionalità di calcolo principali di un CPU. Complessivamente si tratta dell'unione di più circuiti combinatori, ognuno dei quali compie un solo tipo o pochi tipi di calcoli; tale complesso si ottiene attraverso l'uso opportuno di multiplatori per selezionare il risultato a cui si è interessati.

Figura u99.1. Schema esemplificativo di come si possono connettere assieme più componenti per formare una sola ALU: ogni componente riceve una copia dei valori in ingresso, ma l'uscita viene selezionata da un multiplatore, attraverso un codice che rappresenta la funzione da richiedere alla ALU.



Negli esempi, dove possibile, si suddivide il lavoro in moduli da 8 cifre binarie; inoltre, le varie componenti vengono realizzate in modo che possano operare con ranghi differenti di bit: 8, 16, 24 e 32.

Indicatori

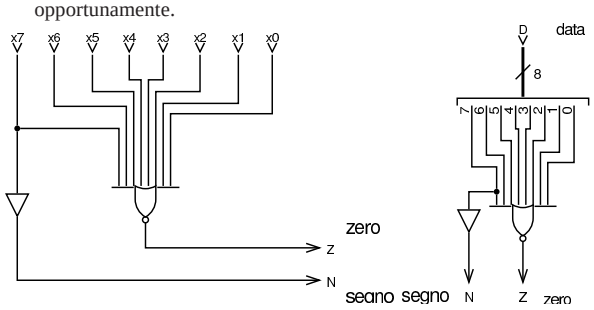
Quando opportuno, devono essere fornite informazioni aggiuntive sui risultati dei calcoli, cosa che si fa normalmente attraverso degli indicatori, costituiti in pratica da delle uscite aggiuntive. Gli indicatori più comuni sono: riporto, zero, segno e traboccamento.

Denominazione	Sigla comune	Descrizione
zero	Z	Indica che il risultato dell'operazione è pari a zero.
segno negativo	N	Indica che, nel risultato, la cifra più significativa è pari a uno, per cui, se la rappresentazione numerica tiene conto del segno, si tratta di un numero negativo.
riporto *carry*	C	Indica che il risultato complessivo dovrebbe avere una cifra in più rispetto a quanto ottenuto, il quale è così troncato della sua cifra più significativa.
traboccamento *overflow*	O	Si manifesta quando il risultato non è valido perché risulta troncato nella parte più significativa, o perché risulta di segno diverso rispetto a quello che dovrebbe avere. Nella somma lo si determina confrontando gli ultimi due riporti e trovando che sono differenti.

La verifica del risultato pari a zero può essere fatta poco prima del-

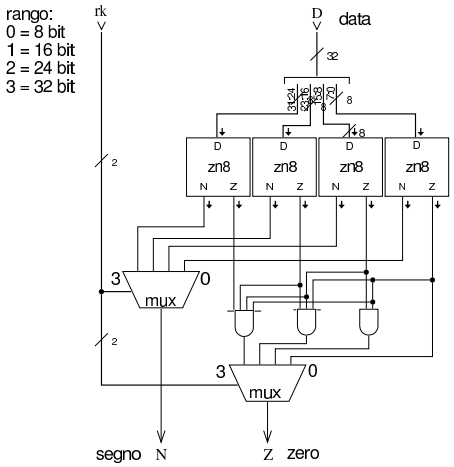
l'uscita dell'ALU, con l'ausilio di una porta NOR, la quale produce un risultato solo quando nessun ingresso è attivo; allo stesso modo, si determina facilmente il segno del risultato (ammesso che il contesto consideri la presenza di un segno), controllando il valore del bit più significativo.

Figura u99.3. Modulo **zn8**: controllo del risultato pari a zero e della presenza eventuale di un segno. A destra appare lo stesso circuito in modo compatto, dove gli otto ingressi sono raccolti opportunamente.



Volendo gestire, a seconda dei casi, interi con rango diverso, si può utilizzare quanto già visto nella figura precedente, in forma di modulo, costruendo un sistema un po' più complesso.

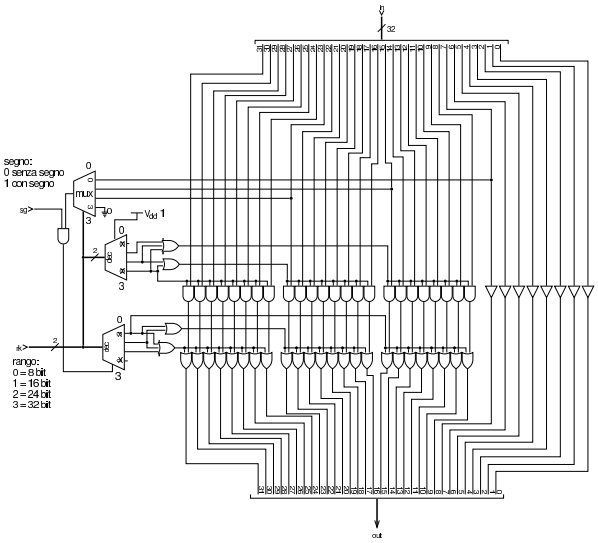
Figura u99.4. Modulo **zn32**: valutazione degli indicatore «zero» e «segno», per interi da 8 a 32 bit. L'ingresso ***rk*** (*rank*) consente di stabilire il rango: si va da zero che rappresenta solo 8 bit, fino a 3 che richiede un rango di 32 bit. Il modulo **zn8** è descritto nella figura u99.3.



Troncamento di un valore in base al rango

I moduli che vengono presentati nelle sezioni successive, consentono di operare su valori a 32 bit, con la possibilità di limitare l'intervento a ranghi inferiori (ma sempre multipli di otto). I risultati che si ottengono sono validi solo nell'ambito del rango selezionato, ma ci possono essere dei contenuti ulteriori, da ignorare, nei bit più significativi oltre al rango voluto. Il modulo successivo, consente di filtrare opportunamente ciò che è al di fuori del rango desiderato, con la possibilità di estendere il segno, in modo da non creare confusione, o comunque per ridurre la possibilità di errori.

Figura u99.5. Modulo **rk**: filtro dei bit al di fuori del rango selezionato, tenendo conto del segno se il valore in ingresso è da intendersi con segno.



Inversione di segno

Per la moltiplicazione e la divisione, si rende necessario un modulo in grado di invertire il segno di un numero binario. Il modulo **minus8** interviene su 8 bit e serve per realizzare moduli di rango maggiore.

Figura u99.6. Modulo **minus8**: inverte il segno di un numero a 8 bit se l'ingresso ***M*** è attivo e lo è anche l'ingresso ***Ci***. Il modulo **ha** è un semiaddizionatore (*half adder*) comune; gli ingressi ***a*** e ***b*** del semiaddizionatore sono intercambiabili.

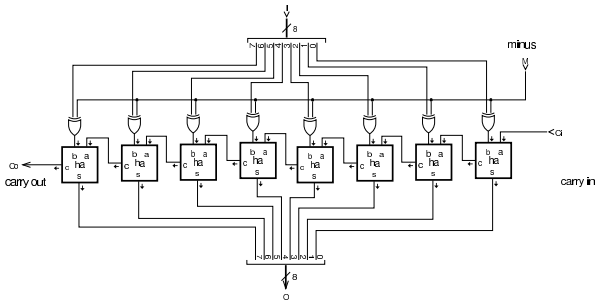
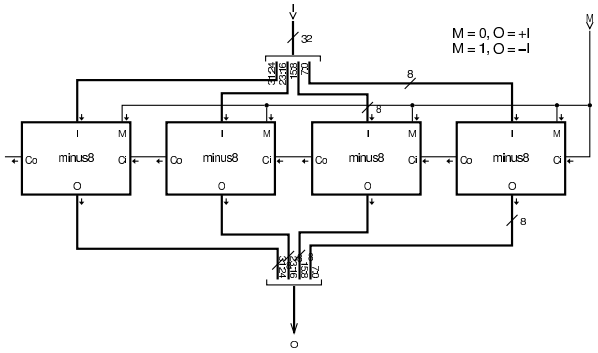


Figura u99.7. Modulo **minus32**: inverte il segno di un numero a 32 bit, se l'ingresso ***M*** è attivo e lo è anche l'ingresso ***Ci***. Il modulo **minus8** è descritto nella figura u99.6.



Somma e sottrazione

La figura successiva mostra un modulo da 8 bit per la somma e la sottrazione, gestendo opportunamente il riporto o la richiesta di prestito di una cifra e fornendo l'informazione sull'eventuale traboccamento, confrontando gli ultimi due riporti.

Figura u99.8. Modulo **as8**: somma e sottrazione di interi a otto cifre binarie. L'ingresso ***f*** (*function*) controlla l'applicazione di una somma o di una sottrazione; l'uscita ***O*** fornisce l'informazione sull'eventuale traboccamento. Il modulo **fa** è un addizionatore completo (*full adder*).

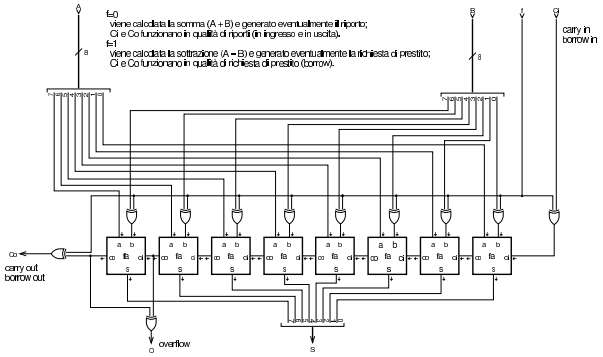
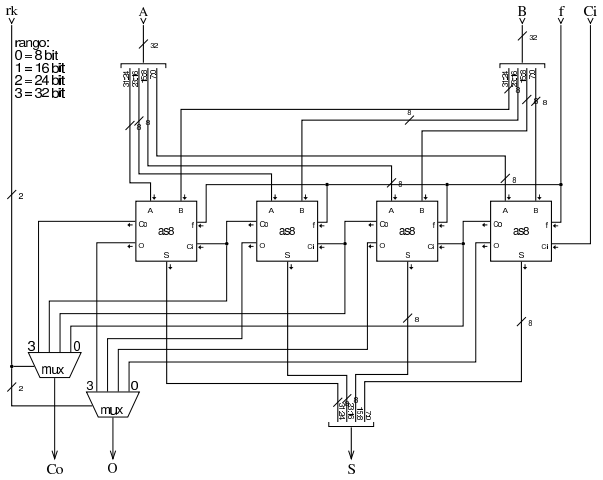


Figura u99.9. Modulo **as32**: somma e sottrazione con la possibilità di controllare il rango dei dati in ingresso e in uscita. Il modulo **as8** è descritto nella figura u99.8.



Scorrimento e rotazione

La figura successiva mostra un modulo per lo scorrimento logico e aritmetico su 8 bit. Gli ingressi e le uscite ***lin***, ***lout***, ***rin***, ***rout*** (*left*/*right in*/*out*), servono a produrre lo scorrimento e anche la rotazione su una scala multipla di 8 bit. Le uscite ***Cl*** e ***Cr*** (*carry left*/*right*) riproducono sempre il valore del bit che viene sospinto all'esterno (dal lato sinistro o dal lato destro), mentre l'uscita ***O*** (*overflow*) si attiva solo in presenza di un traboccamento, dovuto a uno scorrimento aritmetico a sinistra che fa cambiare di segno al valore.

Figura u99.10. Modulo **sh8**, per lo scorrimento logico e aritmetico a 8 bit.

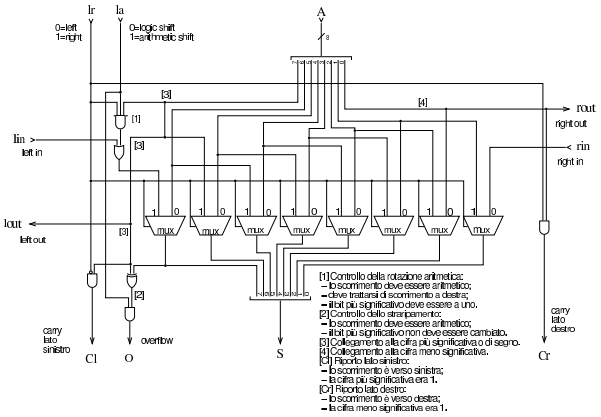


Figura u99.11. Modulo **sh32**: scorrimento logico e aritmetico a 32 bit, con la possibilità di intervenire su un rango inferiore. L'uscita ***C*** (*carry*) restituisce il valore del bit che viene sospinto a sinistra o a destra, in base al contesto. Il modulo **sh8** è descritto nella figura u99.10.

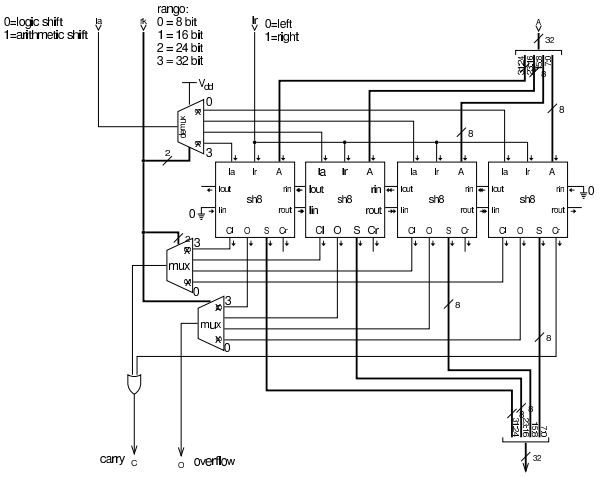


Figura u99.12. Modulo **ro32**: rotazione a 32 bit, con la possibilità di intervenire su un rango inferiore. Dal momento che nella rotazione non si distingue tra una modalità «logica» rispetto a una aritmetica, non è presente un'uscita che possa segnalare lo straripamento. Il modulo **sh8** è descritto nella figura u99.10.

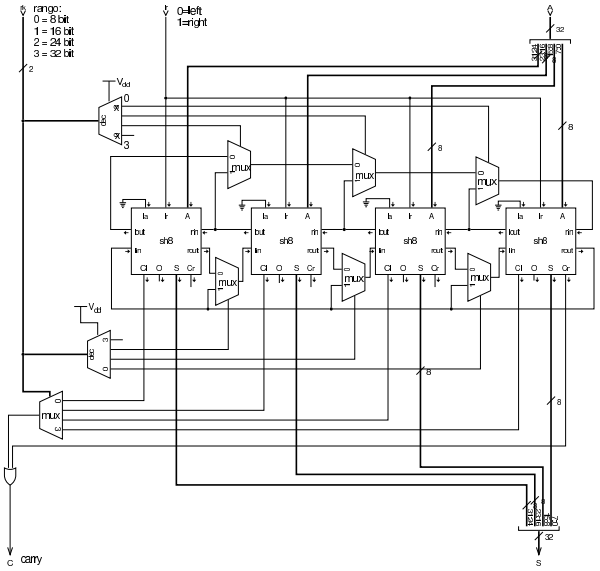
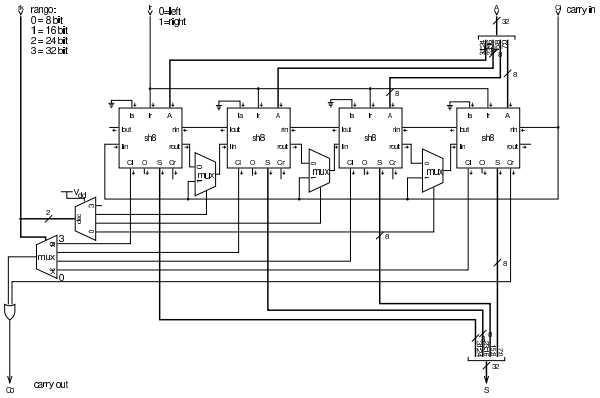


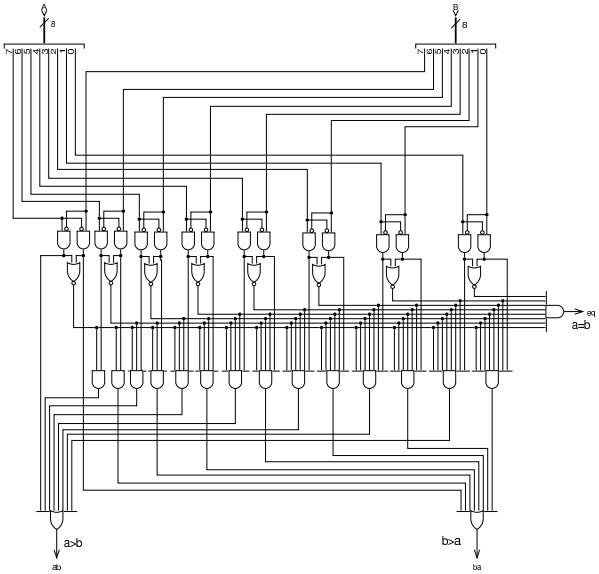
Figura u99.13. Modulo **rc32**: rotazione utilizzando il riporto, in quanto il valore del riporto in ingresso è ciò che viene inserito dal lato opposto alla direzione dello scorrimento. Il modulo **sh8** è descritto nella figura u99.10.



Comparazione di valori

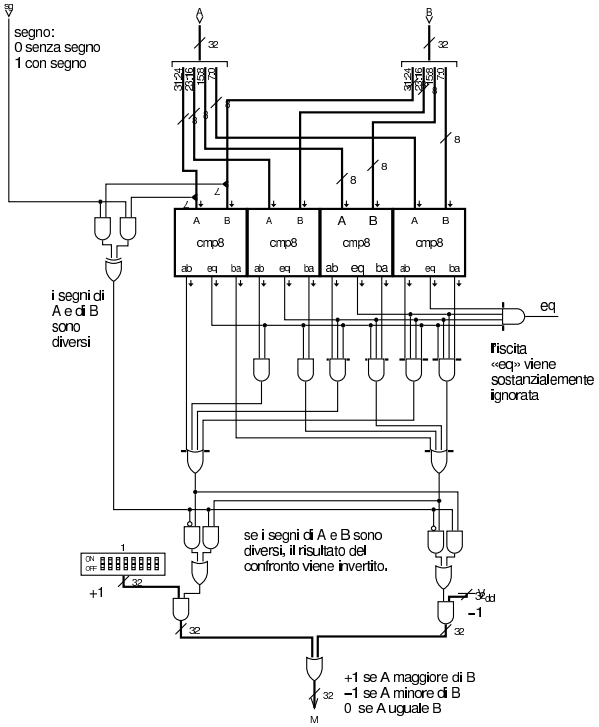
La comparazione tra due valori consente di determinare se questi sono uguali o se uno dei due sia maggiore. Il modulo **cmp8** esegue una comparazione di valori senza segno, attivando un'uscita differente a seconda dei tre casi possibili: $a>b$, $a<b$, $a=b$.

Figura u99.14. Modulo **cmp8**: comparazione di interi, senza segno, a 8 bit.



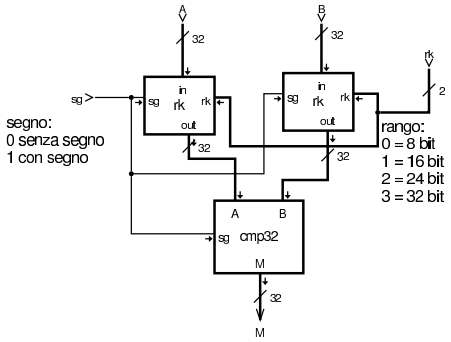
Il modulo **cmp8** può essere utilizzato per il confronto di valori espressi su una quantità multipla di bit; in tal caso, si può introdurre il controllo sul segno: se i valori da confrontare si intendono con segno, se i segni dei due valori sono discordi, l'esito del confronto va invertito. Il modulo **cmp32** confronta due valori a 32 bit, restituendo l'esito del confronto in forma di valore a 32 bit: zero se i valori sono uguali; +1 se $A>B$; -1 se $A<B$.

Figura u99.15. Modulo **cmp32**: confronto a 32 bit, con la possibilità di distinguere tra valori senza segno o con segno. Il modulo **cmp8** è descritto nella figura u99.14.



Il modulo **cmp32** non consente di ridurre il rango di bit preso in considerazione nel confronto; pertanto, se si desidera poter controllare il rango, occorre aggiungere il filtro del modulo **rk** (figura u99.5), come si vede nella figura successiva.

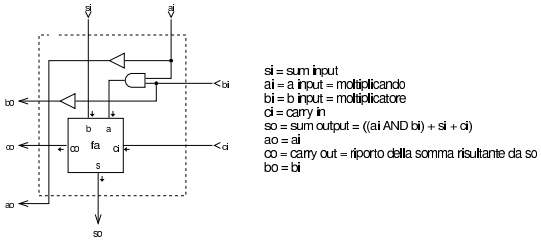
Figura u99.16. Esempio di filtro attraverso il modulo **rk** (figura u99.5).



Moltiplicazione

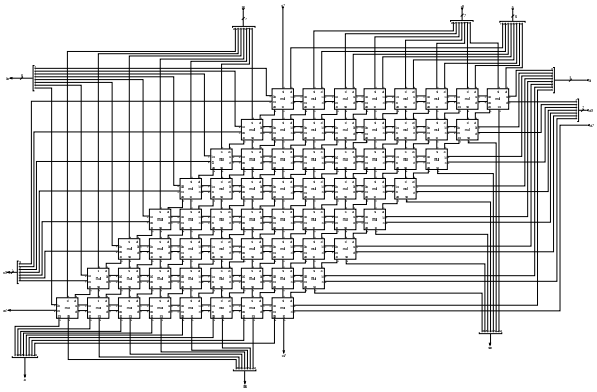
Come già descritto in precedenza, per la moltiplicazione ci si avvale di un modulo apposito che somma il risultato dell'operatore AND sui due valori in ingresso. Viene richiamato nella figura successiva.

Figura u99.17. Modulo **mul**: moltiplicazione a un bit. Questo modulo viene poi usato per la realizzazione di quello a otto bit.



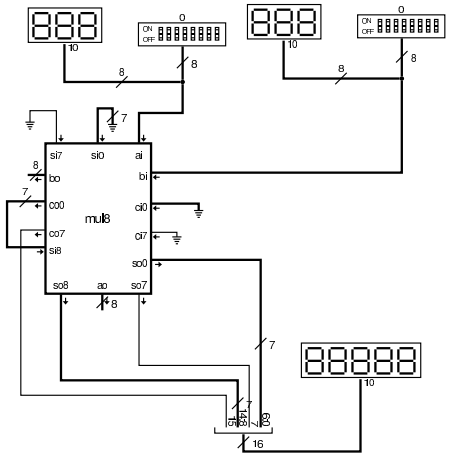
Per la moltiplicazione viene proposto il modulo **mul8**, a 8 bit, con tutte le connessioni necessarie a collegarlo ad altri moduli uguali, per realizzarne di più grandi in forma compatta.

Figura u99.18. Modulo **mul8**: moltiplicazione a 8 bit.



Dato che il modulo **mul8** è abbastanza complesso a causa delle connessioni di cui dispone, nella figura successiva si vede come potrebbe essere utilizzato da solo, per numeri a 8 bit.

Figura u99.19. Utilizzo del modulo **mul8** da solo.



Per procedere gradualmente, si vede anche come potrebbe essere sviluppato un modulo a 16 bit, per il quale servono quattro moduli **mul8**.

Figura u99.20. Modulo **mul16**: moltiplicazione a 16 bit senza segno. Il modulo **mul8** è descritto nella figura u99.18.

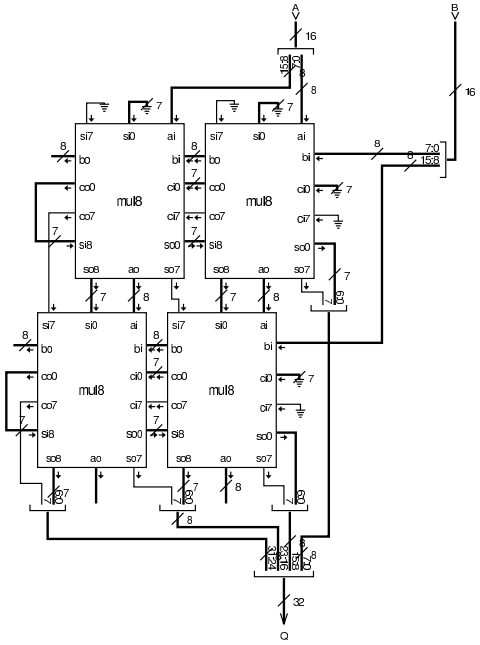
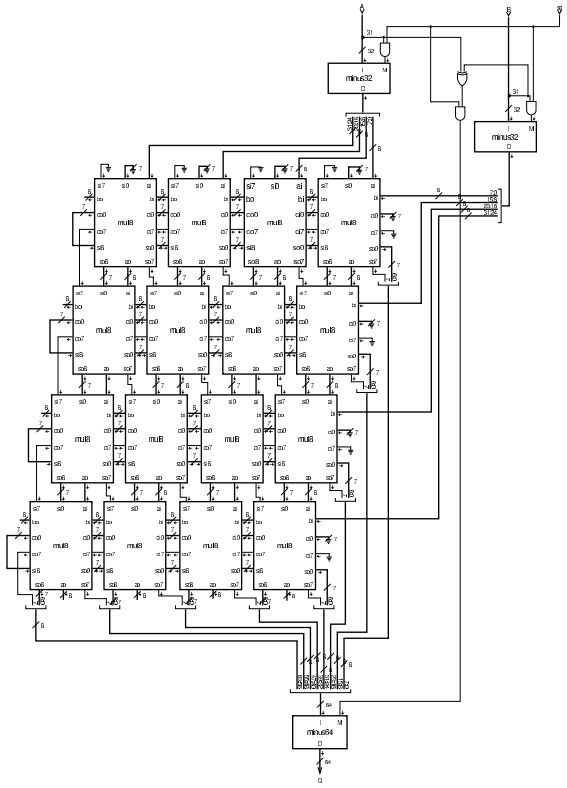


Figura u99.21. Modulo **`mul32`**: moltiplicazione a 32 bit con segno. Il modulo **`mul8`** è descritto nella figura u99.18, mentre i moduli **`minus`**... sono descritti a partire dalla figura u99.6.



Divisione

«

Per la divisione ci si avvale di un modulo che esegue la sottrazione del divisore dal dividendo, in fasi successive. Il modulo in questione è già apparso in precedenza nel capitolo, ma viene ripresentato per maggiore comodità.

Figura u99.22. Modulo **`div`**: componente usato per il calcolo della divisione, attraverso la sottrazione del divisore dal sottraendo. Questo modulo consente di operare su un solo bit. Ci si avvale del modulo **`fs`**, corrispondente a un sottrattore completo (*full subtractor*), descritto nella sezione u0.10.

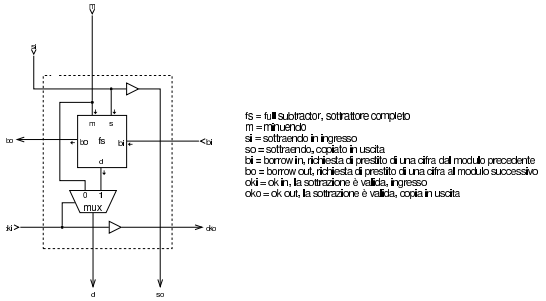


Figura u99.23. Modulo **`div8`**: divisione intera a 8 bit, senza segno. Si utilizzano 64 moduli **`div`** per sottrarre dal dividendo il divisore, fino a trovare il quoziente e il resto. Il modulo è organizzato in modo da potersi connettere con altri moduli uguali e operare su interi aventi un numero maggiore di cifre. Il modulo **`div`** è descritto nella figura u99.22.

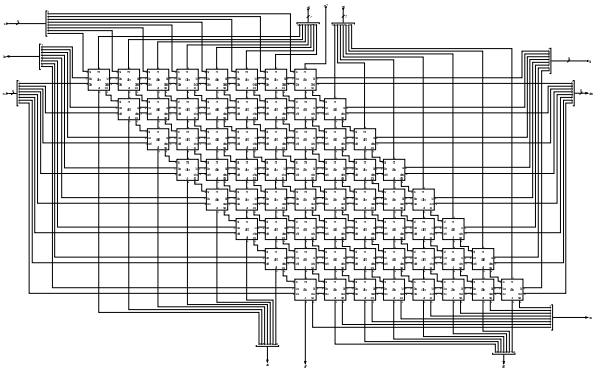


Figura u99.24. Esempio di utilizzo del modulo **`div8`**, per dimostrare in che modo vanno usati i collegamenti di cui è provvisto. Il modulo **`div8`** è descritto nella figura u99.23.

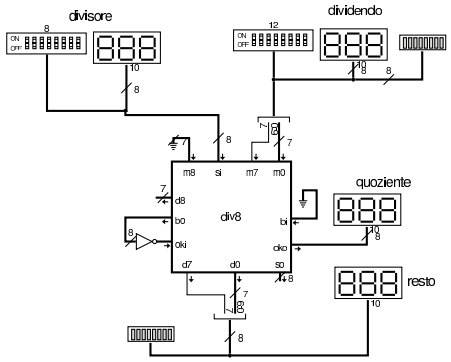


Figura u99.25. Modulo **`div16`**: divisione intera, senza segno, a 16 bit. Si utilizzano quattro moduli **`div8`** collegati opportunamente tra loro. Il modulo **`div8`** è descritto nella figura u99.23.

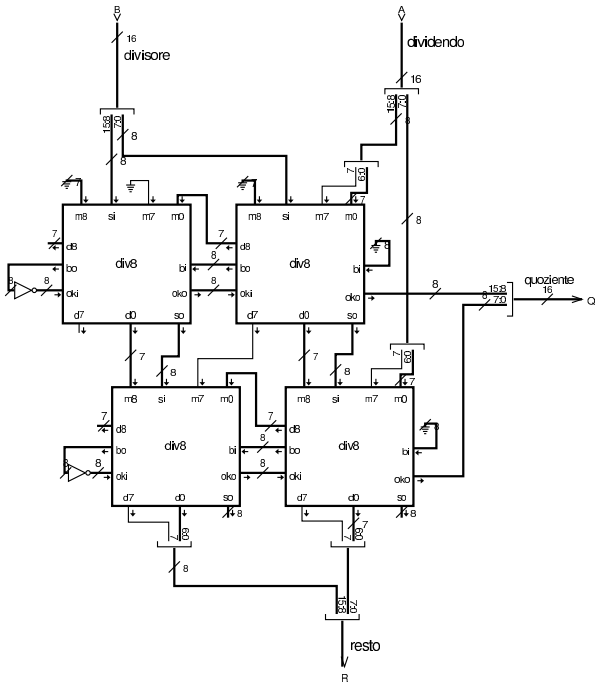
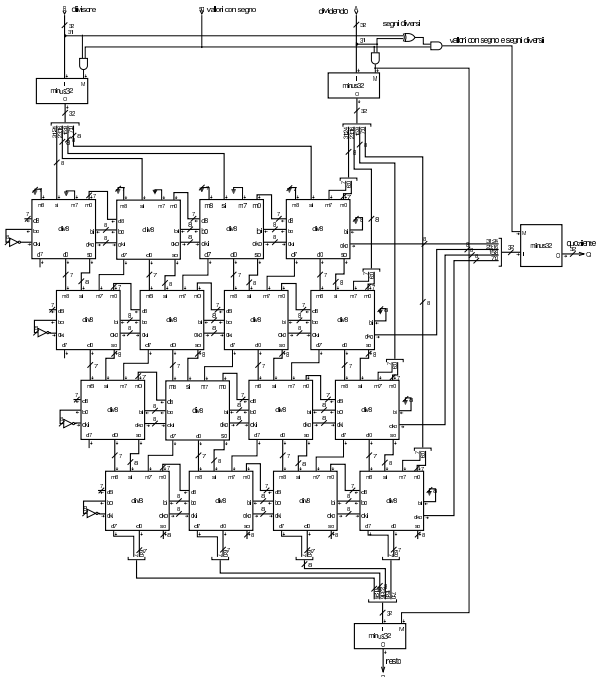


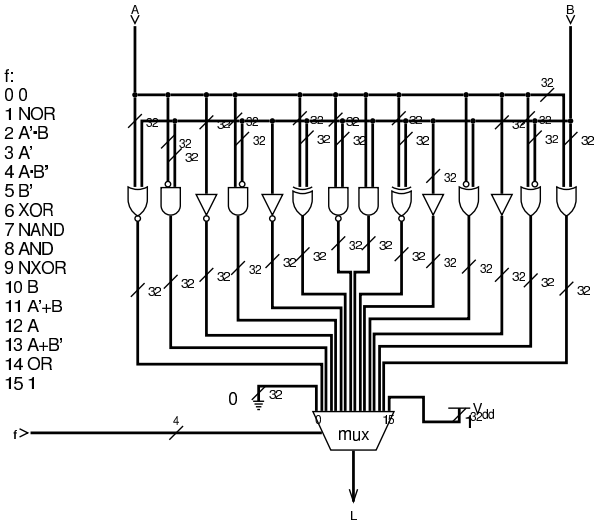
Figura u99.26. Modulo **`div32`**: divisione intera, con o senza segno, a 32 bit. Si utilizzano sedici moduli **`div8`** e quattro moduli **`minus32`** per il controllo del segno, se è richiesto un calcolo che ne tenga conto. Il modulo **`div8`** è descritto nella figura u99.23, mentre il modulo **`minus32`** è descritto nella figura u99.7.



Unità logica

Per unità logica si intende quella che esegue operazioni booleane, bit per bit, su valori interi espressi in forma binaria. Di solito le operazioni disponibili sono poche (AND, OR, NOT, XOR), ma nell'esempio della figura successiva appaiono tutti i casi già descritti nella tabella u98.4, anche se ciò può essere superfluo.

Figura u99.27. Modulo **`logic32`**: realizza le 16 operazioni logiche che si possono ottenere da un circuito combinatorio con due ingressi e un'uscita. In questo caso, ogni componente ne rappresenta in realtà 32, in parallelo: solo l'ingresso ***f*** che serve a selezionare la funzione, è sempre lo stesso.



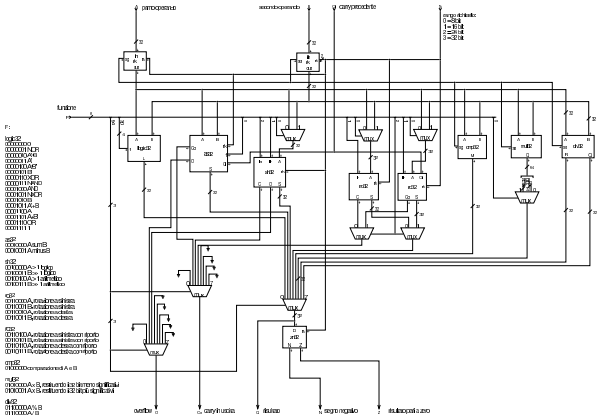
ALU completa

Nella figura successiva viene proposta una, ALU completa di tutti i moduli descritti in questo gruppo di sezioni. Valgono le osservazioni seguenti:

- i moduli della rotazione sono gestiti come se fossero uno solo, in cui la funzione di rotazione con o senza riporto avviene in base alla selezione del tipo di scorrimento; inoltre, sia per lo scorrimento, sia per la rotazione, è possibile scegliere su quale ingresso intervenire;
- per risolvere in qualche modo il problema del risultato della moltiplicazione, che occupa 64 bit, si è scelto di selezionare quale porzione del risultato si vuole ottenere, anche se ciò comporta in pratica un raddoppio del tempo necessario alla moltiplicazione, perché ogni volta il calcolo viene ripetuto;
- i moduli pescano dalla linea dell'ingresso ***F*** (*function*) i bit che gli servono per adeguare il proprio comportamento, tenendo conto che i primi quattro bit di ***F*** servono ai multiplatori che selezionano le uscite da prelevare, mentre i quattro bit più significativi sono quelli che sono affidati ai moduli rispettivi.

Si comprende che si tratta di una soluzione che non è ottimale dal punto di vista delle prestazioni, avendo soltanto uno scopo dimostrativo.

Figura u99.28. Modulo **alu32**: ALU completa a 32 bit. L'ingresso ***F*** determina il comportamento della ALU.



Unità aritmetico-logica e registri espandibili

Unità aritmetico-logica a bit singolo 779
Registri 781

In questa sezione viene presentato uno studio alternativo per la costruzione di una ALU da un solo bit, allineabile in parallelo per ottenere ranghi superiori, assieme a dei registri, anche questi a singolo bit, ma espandibili. Per comprendere il funzionamento e l'uso dei registri, è comunque necessario leggere la descrizione relativa ai flip-flop che appare a partire dalla sezione u101.

Unità aritmetico-logica a bit singolo

Figura u100.1. Modulo **as**: per ***f*** =0, somma con riporto; per ***f*** =1 sottrazione con prestito; se il riporto (o il prestito) in ingresso è diverso da quello in uscita, si verifica lo straripamento.

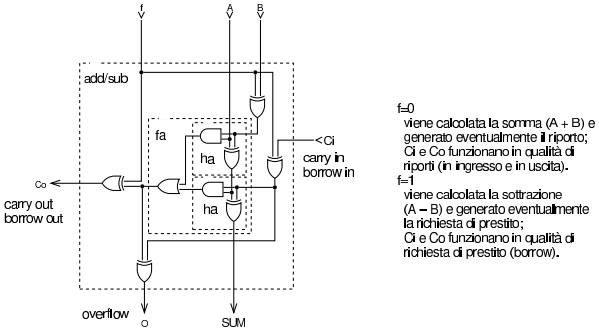


Figura u100.2. Modulo **logic**: unità logica, comandata dall'ingresso ***f*** che seleziona il tipo di operazione logica. Nella didascalia interna dei valori assegnabili all'ingresso ***f***, il simbolo «+» va inteso come operatore OR.

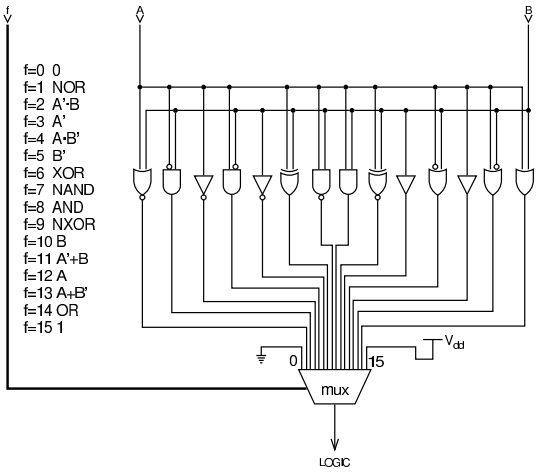


Figura u100.3. Modulo **ltgt**: confronto. Data la necessità di realizzare una ALU espandibile, il confronto avviene a livello di bit, usando il riporto in ingresso in caso di uguaglianza. L'esito viene trasmesso attraverso il riporto in uscita. Il modulo è composto da due parti; nella figura, nell'ordine, appaiono i moduli: **lt** (A<B), **gt** (A>B), **ltgt**. Il modulo complessivo **ltgt** esegue il primo o il secondo confronto, in base al valore contenuto nell'ingresso ***f***. Il controllo di uguaglianza non viene fatto, perché per ottenerlo è sufficiente sfruttare l'unità logica con l'operatore NXOR.

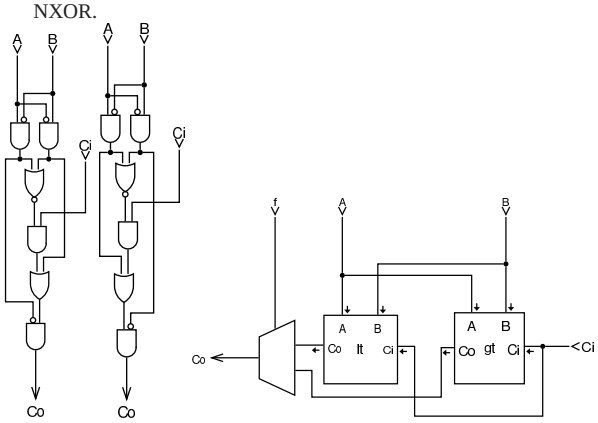


Figura u100.4. Moduli **shl** e **shr**: scorrimento a sinistra e a destra. Lo scorrimento avviene necessariamente utilizzando il riporto; per la precisione, per lo scorrimento a sinistra si usa il riporto «normale», mentre per lo scorrimento a destra si usa un riporto apposito, da sinistra a destra, indicato con le sigle ***Di***, ***Do***. Il riporto a destra non tiene conto del segno, perché in questa fase si considera un solo bit. L'ingresso ***f*** serve, in entrambi i casi, a selezionare l'operando sul quale intervenire: ***A*** o ***B***.

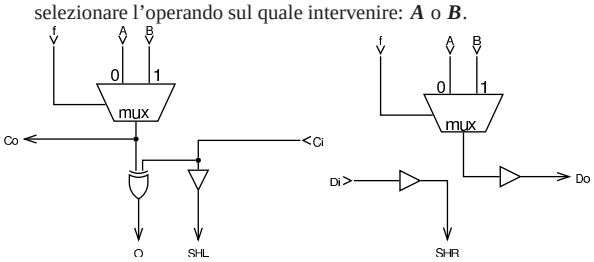


Figura u100.5. Modulo **z**: Zero. Determina se il valore è pari a zero e se lo sono anche i moduli precedenti (attraverso il riporto). Se fino a questa cifra i valori sono a zero, attiva il riporto in uscita. L'ingresso ***f*** consente di selezionare quale operando verificare.

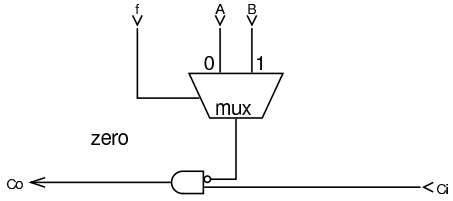


Figura u100.6. Modulo **alu**: ALU completa dei vari componenti aritmetici, logici e di scorrimento. Quando l'ingresso ***f*** ha il bit più significativo pari a zero, contiene la funzione dell'unità logica, altrimenti contiene le funzioni per gli altri moduli. Gli altri moduli, quando contengono a loro volta l'ingresso ***f***, questo lo traggono da f_3 di quello principale, ovvero dal quarto bit.

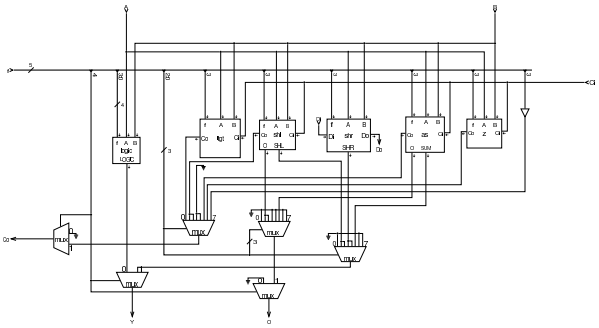
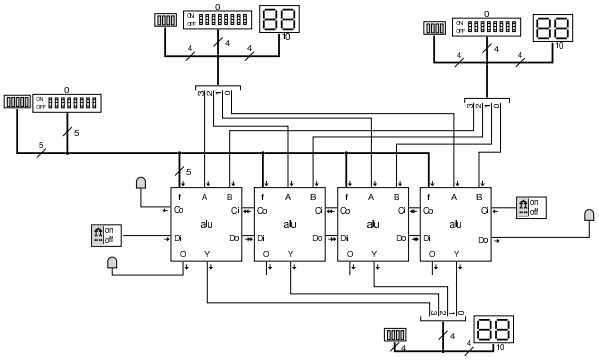


Figura u100.7. Esempio di utilizzo in parallelo del modulo **alu**, per ottenere un'elaborazione a quattro bit. Vanno osservate in particolare le connessioni delle linee di riporto ***C*** e ***D***.



Registri

Attorno alla ALU vengono collocati dei registri, ovvero delle unità di memoria in grado di memorizzare, ognuna, un solo valore. Per i registri si utilizzano flip-flop D, semplici, come nella prima delle figure. I flip-flop sono descritti a partire dalla sezione u101; qui, per comprendere il meccanismo, basti intendere che ricevono l'informazione da memorizzare attraverso l'ingresso ***D***, ma solo quando l'ingresso ***E*** (*enable*) è attivo, quindi mantengono l'informazione disponibile nell'uscita ***Q***.

Figura u100.8. Modulo **d**: Flip-flop D semplice, usato per la realizzazione dei registri.

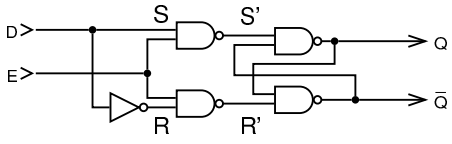


Figura u100.9. Modulo **`ralu`**: ALU e registri, a un solo bit. Gli ingressi ***SRC*** e ***DST***, determinano, rispettivamente, l'origine e la destinazione del movimento di un'informazione, attraverso il bus. In pratica, ***SRC*** seleziona cosa mettere nel bus, e ***DST*** seleziona quale registro deve salvare questa informazione. L'ingresso ***ALU*** contiene la funzione da far svolgere alla ALU, nel caso sia selezionata l'origine corrispondente al terzo registro, che in pratica è l'uscita della ALU stessa. L'ingresso ***E*** va attivato quando tutti gli altri ingressi sono stati configurati correttamente, per il tempo necessario al registro di destinazione per recepire l'informazione contenuta nel bus. Va osservato che la ALU esegue le sue operazioni sulla base del contenuto dei primi due registri, e lo fa continuamente; inoltre, il terzo registro non esiste, in quanto è il riferimento usato per individuare l'uscita della ALU, per cui non è possibile scrivere direttamente in questo terzo registro, senza intervenire con una funzione della ALU. Gli ingressi ***SRC*** e ***DST*** hanno un bit in più, necessario per controllare altri otto registri esterni, necessari per l'accesso alla memoria.

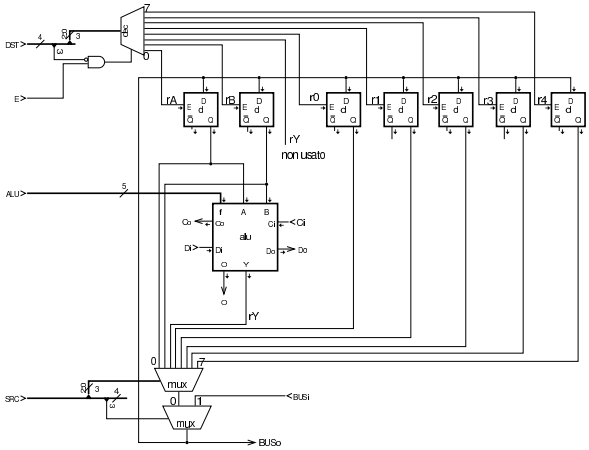


Figura u100.10. Modulo **`ralu4`**: collocazione in serie di quattro moduli a un solo bit. Lo stesso procedimento si seguirebbe per ranghi superiori.

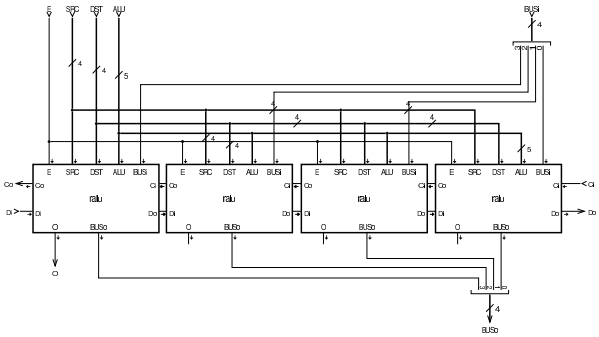
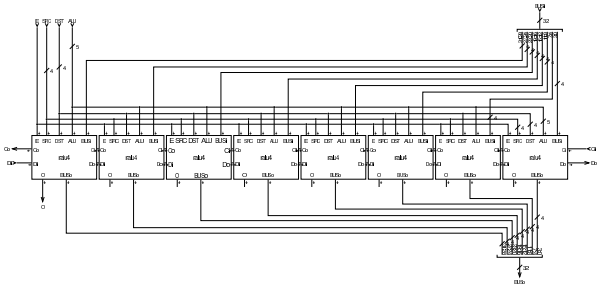


Figura u100.11. Modulo **`ralu32`**: aggregazione di più moduli più piccoli, per un rango più ampio.



Flip-flop

Ritardo di propagazione 783
Tabelle di verità 784
Flip-flop SR elementare 784
Interruttori senza rimbalzi 787
Flip-flop SR con gli ingressi controllati 787
Flip-flop SR sincrono «edge triggered» 788
Tempo: *setup*/*hold* e *recovery*/*removal* 789
Flip-flop D 790
Flip-flop T 791
Flip-flop JK 792

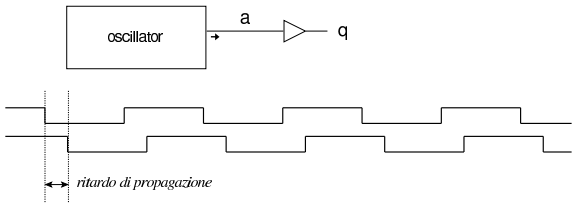
I circuiti combinatori hanno la caratteristica di fornire un certo valore di uscita, unicamente sulla base dei valori presenti in ingresso, anche se la formazione del valore di uscita richiede comunque un piccolo lasso di tempo, rispetto alla disponibilità dei dati di partenza. Pertanto, si dice che i circuiti combinatori non hanno memoria, in quanto non tengono conto di ciò che è avvenuto in precedenza. Al contrario, un ***circuito sequenziale***, tiene conto della dinamica con cui provengono i dati in ingresso, mantenendo uno ***stato***, dal quale dipendono gli effetti dei cambiamenti successivi dei dati in ingresso.

I circuiti sequenziali si basano su componenti noti come ***memorie***, le quali si realizzano attraverso i ***flip-flop***. All'interno della categoria delle memorie, la documentazione scientifica e tecnica tende a usare il termine ***latch*** che si riferisce alla proprietà di questi circuiti di «scattare» da uno stato a un altro, riservando la definizione di flip-flop solo a un insieme ristretto di tali circuiti, in quanto avente caratteristiche più sofisticate. Tuttavia, qui si preferisce usare la qualifica di flip-flop per tutti questi circuiti, distinguendo di volta in volta il livello di sofisticazione di ogni variante.

Ritardo di propagazione

In un circuito combinatorio comune, perché l'uscita si adegui ai cambiamenti avvenuti in ingresso, si richiede un piccolo intervallo di tempo, noto come ***ritardo di propagazione***. Questo ritardo dipende dalle caratteristiche fisiche e meccaniche del dispositivo con cui si realizza effettivamente il circuito combinatorio.

Figura u101.1. Esempio di ritardo nell'attraversamento di un *buffer*, ovvero di una porta logica non invertente: come si intende dallo schema, l'ingresso è pilotato da un oscillatore che produce un segnale alterno e nell'uscita si ottiene lo stesso segnale, ma leggermente ritardato.



Il ritardo di propagazione si può sfruttare per ottenere un impulso molto breve, come si vede nella figura successiva.

Figura u101.2. Esempio di impulso generato sfruttando il ritardo di propagazione. In questo schema si intende che gli ingressi della porta AND siano bilanciati correttamente, nel senso che il cambiamento di stato da un ingresso o dall'altro, si rifletta sull'esito restituito nello stesso tempo.

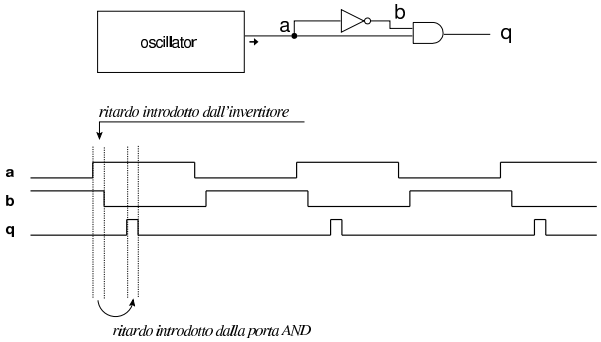
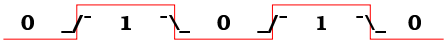


Tabelle di verità

La tabella di verità è un modo con cui si rappresenta il comportamento di un circuito combinatorio, per cui si mostrano i valori in uscita come funzione dei valori in entrata, senza contare il ritardo di propagazione e la dinamica con cui si formano i dati in ingresso. Trattando invece di circuiti sequenziali, le tabelle di verità, ammesso che si utilizzino, vanno interpretate di volta in volta in base al contesto particolare. Nelle sezioni successive si introducono i flip-flop e nelle tabelle di verità si vuole considerare negli ingressi anche la variazione di stato; pertanto si usa la simbologia seguente:

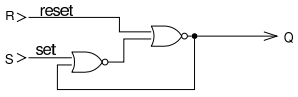
0	zero stazionario
_/‾	variazione da zero a uno
1	uno stazionario
‾_	variazione da uno a zero



Flip-flop SR elementare

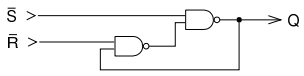
L'esempio più semplice di circuito sequenziale che mantiene la memoria del proprio stato, è quello che si può vedere nelle due figure successive, realizzato rispettivamente con porte NOR e NAND. Nel caso del circuito con porte NOR, si parte da uno stato iniziale in cui l'uscita ***Q*** ha un valore indeterminabile (potrebbe essere zero oppure uno, in base a fattori imponderabili), quindi, attivando l'ingresso ***S*** (*set*), anche solo con un breve impulso, l'uscita ***Q*** si attiva e rimane attiva fino a quando non riceve un impulso dall'ingresso ***R*** (*reset*); nel caso del circuito con porte NAND, gli ingressi ***S'*** e ***R'*** funzionano in modo negato.

Figura u101.5. Circuito elementare di tipo *set-reset* con porte NOR.



S	R	Q
0	0	*invariato*
0	_/‾	0
0	1	0
0	‾_	0
_/‾	0	1
_/‾	_/‾	0
_/‾	1	0
_/‾	‾_	1
1	0	1
1	_/‾	0
1	1	0
1	‾_	1
‾_	0	1
‾_	_/‾	0
‾_	1	0
‾_	‾_	*non ammissibile!*

Figura u101.6. Circuito elementare di tipo *set-reset* con porte NAND.

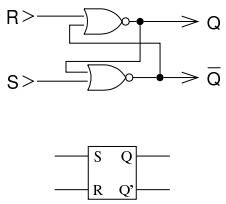


S'	R'	Q
0	0	1
0	_/‾	1
0	1	1
0	‾_	1
_/‾	0	0
_/‾	_/‾	*non ammissibile!*
_/‾	1	1
_/‾	‾_	0
1	0	0
1	_/‾	0
1	1	*invariato*
1	‾_	0
‾_	0	1
‾_	_/‾	1
‾_	1	1
‾_	‾_	1

Nelle tabelle di verità che appaiono a fianco dei disegni, si può osservare che esiste una situazione inammissibile: nel primo caso quando entrambi gli ingressi passano da uno a zero; nel secondo, quando passano da zero a uno. Se si tentasse di farlo, si metterebbe il circuito in risonanza, come viene spiegato meglio in seguito.

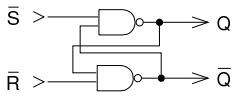
Le ultime due figure apparse rappresentano in realtà i flip-flop SR elementari, i quali però si disegnano solitamente aggiungendo un'uscita ulteriore, ***Q'***, che assume un valore inverso rispetto a ***Q***.

Figura u101.7. Flip-flop SR elementare, realizzato con porte NOR.



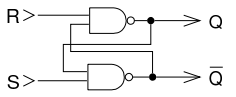
S	R	Q	Q'
0	0	*invariato*	
0	_/‾	0	1
0	1	0	1
0	‾_	0	1
_/‾	0	1	0
_/‾	_/‾	0	1
_/‾	1	0	1
_/‾	‾_	1	0
1	0	1	0
1	_/‾	0	1
1	1	0	0
1	‾_	1	0
‾_	0	1	0
‾_	_/‾	0	1
‾_	1	0	1
‾_	‾_	*non ammissibile!*	

Figura u101.8. Flip-flop SR elementare, realizzato con porte NAND.



S'	R'	Q	Q'
0	0	1	1
0	_/‾	1	0
0	1	1	0
0	‾_	1	0
_/‾	0	0	1
_/‾	_/‾	*non ammissibile!*	
_/‾	1	1	0
_/‾	‾_	0	1
1	0	0	1
1	_/‾	0	1
1	1	*invariato*	
1	‾_	0	1
‾_	0	1	0
‾_	_/‾	1	0
‾_	1	1	0
‾_	‾_	1	0

Figura u101.9. Flip-flop SR elementare, realizzato con porte NAND cambiando il significato degli ingressi.

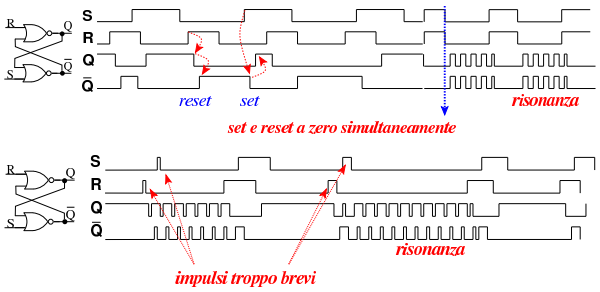


S	R	Q	Q'
0	0	1	1
0	_/‾	0	1
0	1	0	1
0	‾_	0	1
_/‾	0	1	0
_/‾	_/‾	*non ammissibile!*	
_/‾	1	0	1
_/‾	‾_	1	0
1	0	1	0
1	_/‾	1	0
1	1	*invariato*	
1	‾_	1	0
‾_	0	1	0
‾_	_/‾	0	1
‾_	1	0	1
‾_	‾_	1	0

Il flip-flop SR elementare realizzato con porte NAND è equivalente a quello realizzato con porte NOR, se al primo si invertono gli ingressi; tuttavia, il flip-flop SR con porte NAND si usa spesso cambiando nome agli ingressi, come avviene nell'ultima figura mostrata, ma in tal caso, il comportamento non risulta uguale a quello fatto con porte NOR, anche se vi si avvicina. Per questo problema, quando si disegna un flip-flop SR elementare come scatola, bisogna chiarire a quale tabella di verità si sta facendo riferimento; tuttavia, l'uso

di flip-flop SR elementari è molto limitato, pertanto nel disegno dei circuiti è meglio evitare di rappresentarli come scatole.

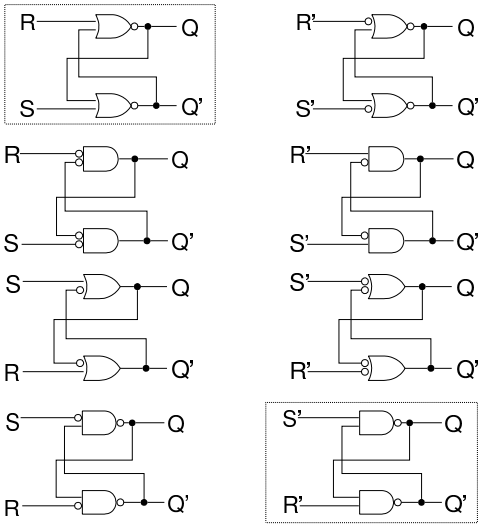
Figura u101.10. Tracciati di un flip-flop SR elementare basato su porte NOR, evidenziando la situazione critica che si crea quando entrambi gli ingressi si trovano attivati assieme e poi si azzerano simultaneamente e quella che si crea quando l'attivazione degli ingressi ha una durata troppo breve.



La figura mostra il tracciato dei valori delle entrate e quello delle uscite di un flip-flop SR elementare e con questo si può vedere l'effetto del ritardo di propagazione, ma soprattutto ciò che accade quando gli ingressi da attivi passano simultaneamente a zero: a causa del ritardo di propagazione, le due uscite rimangono per un po' a zero, poi, però, trovando gli ingressi a zero, si attivano simultaneamente e innescano un circolo vizioso. Pertanto, nel flip-flop SR elementare, è necessario evitare la condizione inammissibile già mostrata nelle tabelle di verità. Ma anche un impulso troppo breve in uno degli ingressi può procurare un effetto di risonanza: in generale, la durata minima di un impulso negli ingressi deve essere tale da consentire al flip-flop di cambiare stato, quando tale impulso lo prevederebbe.

Nonostante la sua limitazione, a proposito della presenza di condizioni di ingresso inammissibili, il flip-flop SR elementare costituisce la base per tutti gli altri tipi di flip-flop. Per tale motivo è importante conoscere in quali altre forme può essere realizzato, come dimostrato nella figura successiva.

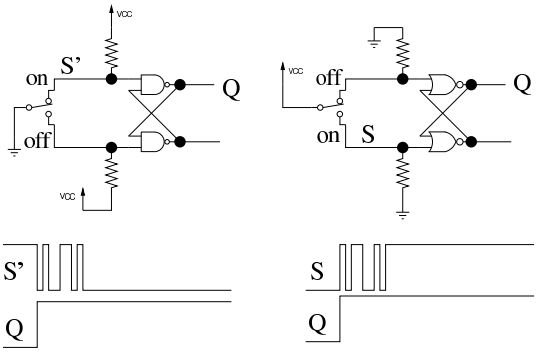
Figura u101.11. Flip-flop SR elementare in varie forme alternative, nelle quali occorre fare attenzione all'ordine degli ingressi. Nella colonna di sinistra si mostrano nella versione a ingressi positivi, mentre in quella destra appaiono nella versione a ingressi invertiti. Nei due gruppi sono evidenziati i due tipi più comuni: NOR e NAND.



Interruttori senza rimbalzi

Nella realizzazione pratica di circuiti logici (elettronici) si ha spesso la necessità di utilizzare degli interruttori o pulsanti. Ma nella vita reale, tali componenti hanno il problema dei rimbalzi, nel senso che l'apertura o la chiusura di un interruttore comporta la creazione di impulsi indesiderabili. Per impedire che tali impulsi arrivino a un circuito, si utilizzano i flip-flop SR elementari, per esempio nella modalità che si può vedere nella figura successiva.

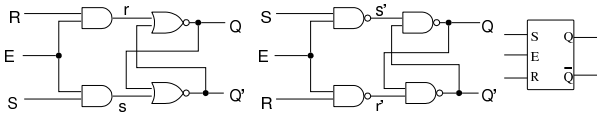
Figura u101.12. Utilizzo di un flip-flop SR elementare per filtrare i rimbalzi di un interruttore.



Flip-flop SR con gli ingressi controllati

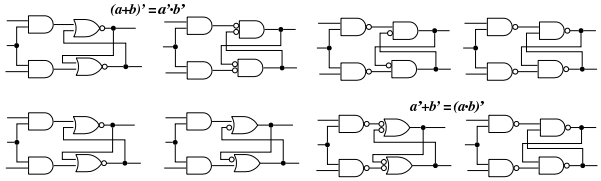
Il flip-flop SR elementare può essere esteso aggiungendo un controllo di abilitazione degli ingressi, con due porte AND o NAND, a seconda del tipo: in pratica, così facendo, quando l'ingresso di abilitazione è attivo, il flip-flop SR funziona normalmente, mentre diversamente è come se entrambi gli ingressi si trovassero a zero.

Figura u101.13. Flip-flop SR controllato: quando l'ingresso ***E*** (*enable*) è attivo, funziona come un flip-flop SR elementare; se invece l'ingresso ***E*** non è attivo, è come se gli ingressi ***S*** e ***R*** fossero disattivati a loro volta. Il circuito appare nelle due versioni più comuni, assieme alla simbologia usata normalmente per rappresentare questo tipo di flip-flop.



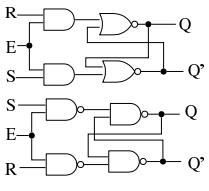
I due schemi che appaiono nella figura sono equivalenti e lo si dimostra facilmente, con l'aiuto dei teoremi di De Morgan, come si vede nei disegni seguenti.

Figura u101.14. Dimostrazione dell'equivalenza dei due modi di rappresentare i flip-flop SR controllati.



Nel flip-flop SR controllato rimangono i problemi di innesco della risonanza già descritti: in pratica, è indispensabile che l'ingresso di abilitazione (***E***) sia attivo solo quando gli ingressi ***S*** e ***R*** si trovano in una condizione valida, ma rispetto al flip-flop SR elementare, non è nemmeno ammesso che gli ingressi siano attivi simultaneamente, perché all'abbassarsi della linea di abilitazione si creerebbe inevitabilmente l'innesco.

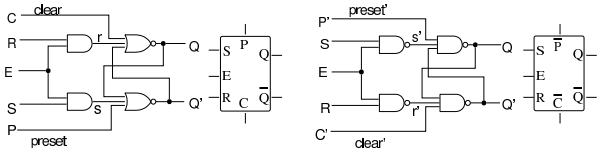
Figura u101.15. Tabella di verità per il flip-flop SR controllato, limitatamente ai casi più significativi.



E	S	R	Q	Q'
0	x	x	*invariato*	
↓	0	0	*invariato*	
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	*non ammissibile!*	
⋮	⋮	⋮		
1	↓	↓	*non ammissibile!*	
↓	1	1	*non ammissibile!*	

Quando si mette in funzione un flip-flop, lo stato delle uscite è indeterminabile. Per poter inizializzare il flip-flop SR controllato, è necessario estendere gli ingressi delle porte del flip-flop SR elementare, come mostrato nella figura successiva. Va osservato che a seconda di come si realizza il flip-flop SR, può darsi che per inizializzare il flip-flop possa richiedersi un valore a uno o a zero.

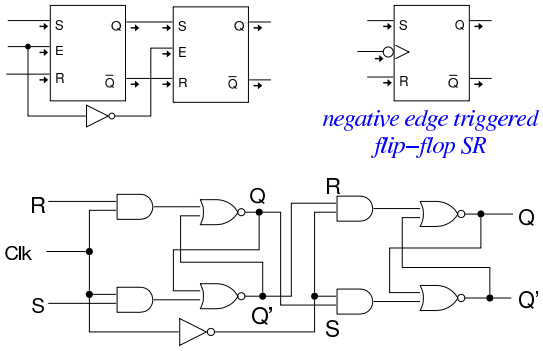
Figura u101.16. Flip-flop SR controllato, con ingressi di inizializzazione: quando l'ingresso ***P*** (*preset*) è attivo, si forza l'attivazione dell'uscita ***Q***; quando l'ingresso ***C*** (*clear*) è attivo, si forza l'attivazione dell'uscita ***Q'***. Il circuito viene mostrato nelle due varianti realizzative comuni, assieme alla rappresentazione simbolica complessiva. Va osservato che nella seconda modalità, gli ingressi ***P*** e ***C*** sono invertiti (negati).



Flip-flop SR sincrono «edge triggered»

Nel flip-flop controllato attraverso l'ingresso di abilitazione (o di *clock*), quando tale ingresso di abilitazione è attivo, i valori degli altri ingressi possono cambiare e il loro cambiamento può trasmettersi regolarmente nel flip-flop modificando eventualmente lo stato delle uscite. Per fare in modo che il controllo di abilitazione fotografi la situazione degli ingressi, occorre che sia pilotato da un impulso abbastanza breve, ma non troppo, tale per cui in quel lasso di tempo i valori degli ingressi non possano cambiare. Per ovviare a questo problema, si possono mettere due flip-flop SR controllati in cascata (*master-slave*), dove il secondo riceve il segnale di abilitazione invertito rispetto al primo; tuttavia, in tal caso l'aggiornamento del flip-flop complessivo si ottiene nel momento in cui il segnale di abilitazione si disattiva, ovvero in corrispondenza del margine negativo (*negative edge*).

Figura u101.17. Flip-flop SR sincrono a margine negativo: quando il segnale di abilitazione passa da attivo a zero, il flip-flop si aggiorna. Il circuito viene mostrato a blocchi e nelle due varianti realizzative comuni, assieme alla rappresentazione simbolica complessiva, dove va osservato che l'ingresso di abilitazione viene annotato con un triangolo, per sottolineare il fatto che il segnale viene recepito in corrispondenza della sua variazione.



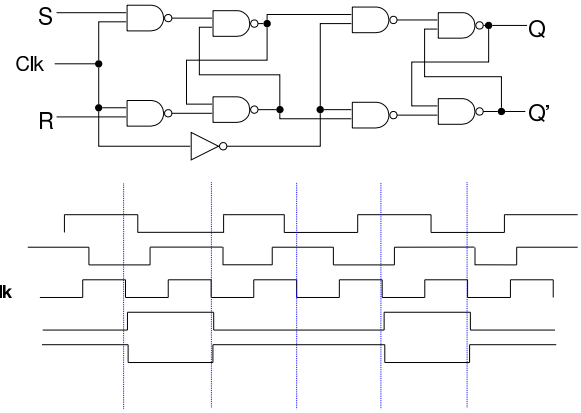


Figura u101.18. Completamento del flip-flop SR sincrono a margine negativo, con gli ingressi di inizializzazione.

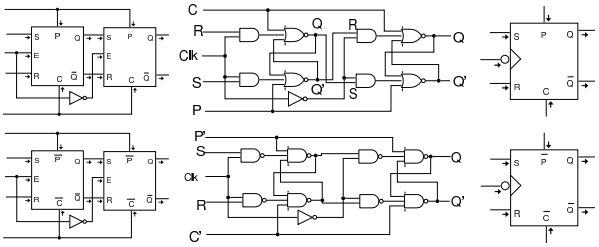
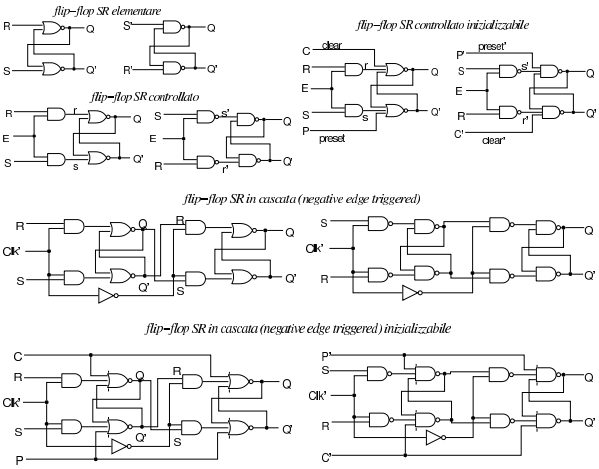


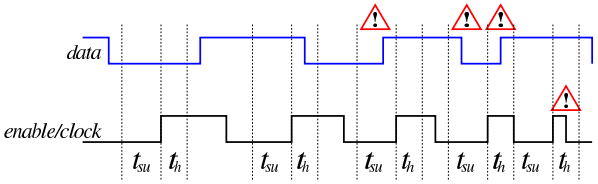
Figura u101.19. Riassunto delle varie tipologie di flip-flop SR.



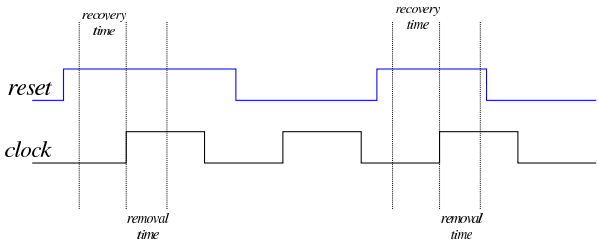
Tempo: *setup*/*hold* e *recovery*/*removal*

Si distinguono due intervalli di tempo significativi per i componenti sincroni, ovvero quelli che hanno un ingresso dati controllato da un ingresso di abilitazione o di *clock*. Si tratta del tempo di attivazione, *setup time*, noto con la sigla t_{su}, e del tempo di mantenimento, *hold time*, noto con la sigla t_h.

Figura u101.20. Esempio di situazioni corrette e non corrette, relativamente ai vincoli del tempo di attivazione (t_{su}) e del tempo di mantenimento (t_h).



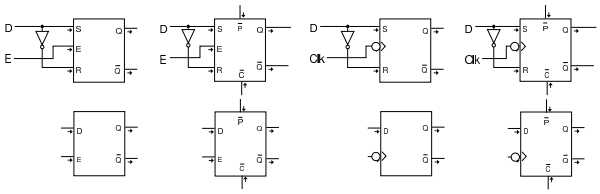
In contesti diversi, quando si vuole sottolineare il fatto che il dato in ingresso è un'informazione asincrona, si usa un'altra terminologia: *recovery time* e *removal time*.



Flip-flop D

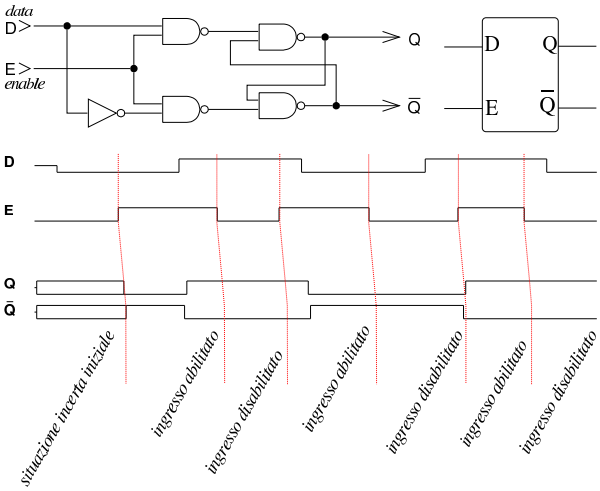
Il flip-flop D (*data*) si ottiene da un flip-flop SR, collegando assieme i due ingressi, invertendo però l'ingresso ***R***. In pratica, il flip-flop D è utile solo quando c'è almeno il controllo di abilitazione degli ingressi.

Figura u101.22. Flip-flop D ottenuto dal flip-flop SR, nelle varie tipologie ammissibili, a confronto con i simboli corrispondenti. Gli ingressi di inizializzazione e di clock sono stati mostrati solo nella versione negata, essendo la più comune.



Sul flip-flop D valgono le stesse considerazioni fatte sul flip-flop SR, per quanto riguarda il tempo di attivazione e il tempo di mantenimento dell'impulso di abilitazione. Rispetto al flip-flop SR, essendoci un solo ingresso dati, non ci sono combinazioni inammissibili.

Figura u101.23. Flip-flop D con ingresso di abilitazione semplice, realizzato utilizzando porte NAND. Quando l'ingresso ***E*** è attivo, il circuito recepisce il dato dall'ingresso ***D*** e lo riproduce attraverso l'uscita ***Q*** (invertendolo nell'uscita ***Q'***).



Il flip-flop D, attivato dalla variazione del segnale di clock, può essere realizzato in cascata, come già visto per il flip-flop RS, Tuttavia esiste un circuito alternativo più efficiente: in tal caso, il margine di attivazione del flip-flop D, diventa quello positivo.

Figura u101.24. Flip-flop D in cascata e nella sua realizzazione classica. Il tracciato riguarda la seconda versione che risulta funzionare a margine positivo.

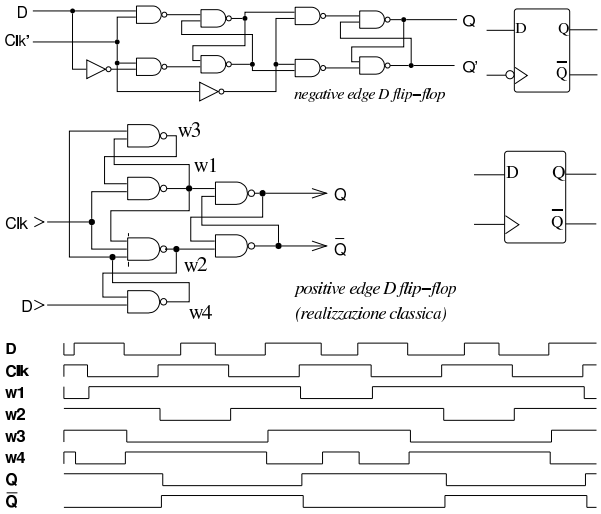
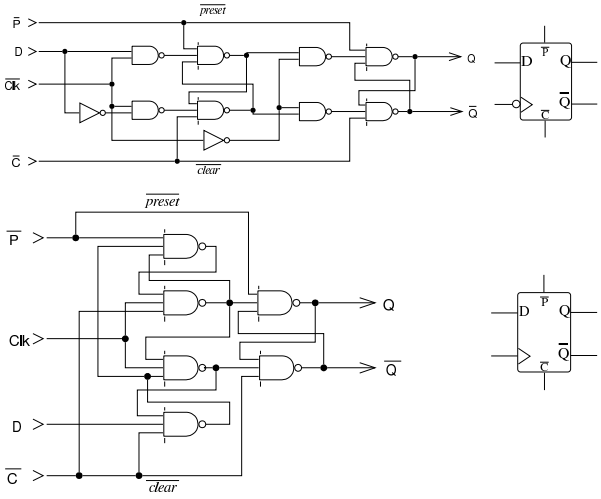


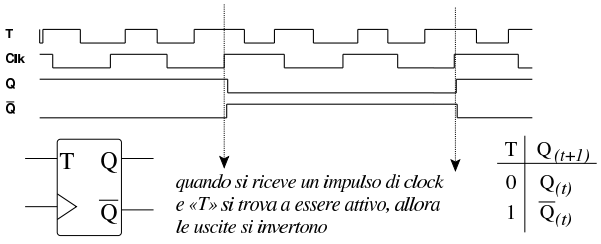
Figura u101.25. Flip-flop D in cascata e nella sua realizzazione classica, con gli ingressi di inizializzazione.



Flip-flop T

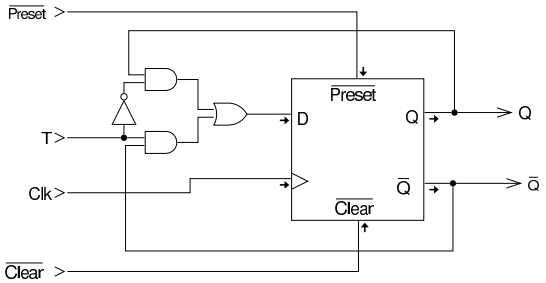
Estendendo leggermente un flip-flop D a margine positivo, è possibile ottenere un flip-flop T (*toggle*), il quale ha lo scopo di invertire il valore delle uscite quando l'ingresso ***T*** si attiva per la presenza di un impulso di *clock*. Nella tabella della verità si usa la notazione $\boldsymbol{Q}_{(t)}$ per indicare il valore dell'uscita ***Q*** nel momento ***t*** e la notazione $\boldsymbol{Q}_{(t+1)}$ per indicare il valore dell'uscita ***Q*** nel momento successivo ***t+1***, corrispondente all'impulso di *clock* successivo.

Figura u101.26. Tracciato, simbolo del flip-flop T e tabella della verità.



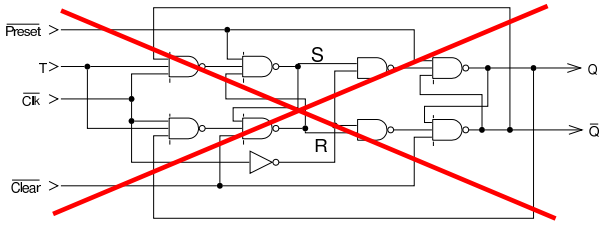
Per realizzare il flip-flop T è necessario tenere in considerazione il fatto che inizialmente non si conosce lo stato delle uscite, quindi è importante poter azzerare il flip-flop all'avvio. Dal momento che si tratta di estendere il flip-flop D, nella figura successiva si parte da quello che dispone degli ingressi di azzeramento e di impostazione.

Figura u101.27. Realizzazione del flip-flop T, partendo da un flip-flop D, munito di ingressi di azzeramento e di impostazione. Questa realizzazione è da preferire rispetto a quella della figura successiva.



La figura successiva mostra una realizzazione diversa del flip-flop T che in sostanza deriva da un flip-flop JK descritto nel libro *Digital Computer Electronics* di Malvino e Brown, nel capitolo dedicato ai flip-flop. Tuttavia, attraverso l'uso di un simulatore, si determina che se l'ingresso ***T*** riceve un impulso che termina prima che ci sia la variazione negativa del clock, si ottiene ugualmente lo scambio dei valori nelle uscite, mentre ciò non dovrebbe avvenire secondo lo schema di funzionamento previsto per un flip-flop T.

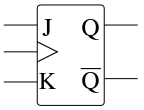
Figura u101.28. Realizzazione del flip-flop T, modificando un flip-flop SR in cascata, fatto a sua volta con porte NAND. In tal caso, lo scambio dei valori di uscita avviene in corrispondenza della variazione negativa dell'impulso di clock (*negative edge triggered*), ma bisogna fare attenzione all'ingresso ***T***, la cui attivazione viene recepita anche se si azzera prima della variazione negativa del clock.



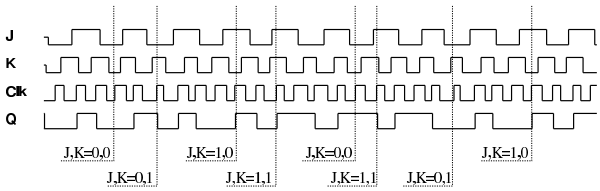
Flip-flop JK

Un'altra variante del flip-flop D, simile al flip-flop T, ma che richiama il funzionamento del flip-flop SR, è il flip-flop JK. Come per il flip-flop T, nella tabella della verità si usa la notazione $Q_{(t)}$ per indicare il valore dell'uscita ***Q*** nel momento ***t*** e la notazione $Q_{(t+1)}$ per indicare il valore dell'uscita ***Q*** nel momento successivo ***t+1***, corrispondente all'impulso di *clock* successivo.

Figura u101.29. Simbolo del flip-flop JK, tabella della verità e tracciato. Si osservi che gli ingressi ***J*** e ***K*** si comportano in modo analogo a gli ingressi ***S*** e ***R*** di un flip-flop SR, con la differenza che l'attivazione di entrambi provoca solo lo scambio dei valori delle uscite, senza altre complicazioni.

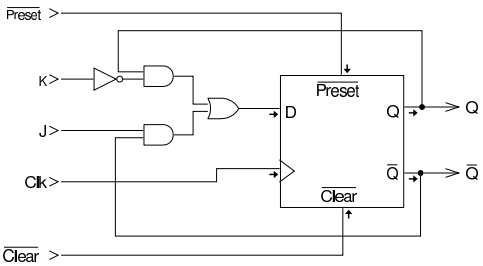


J	K	$Q_{(t+1)}$	
0	0	$Q_{(t)}$	*invariato*
0	1	0	*reset*
1	0	1	*set*
1	1	$\overline{Q}_{(t)}$	*scambio dei valori*



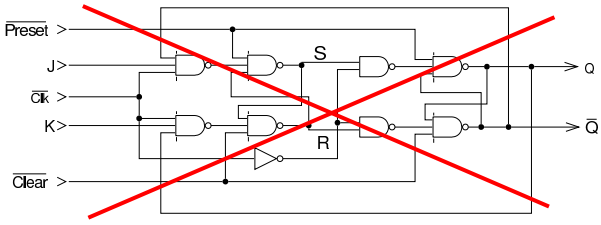
Per realizzare il flip-flop JK è necessario tenere in considerazione il fatto che inizialmente non si conosce lo stato delle uscite, quindi è importante poter azzerare il flip-flop all'avvio. Dal momento che si tratta di estendere il flip-flop D, nella figura successiva si parte da quello che dispone degli ingressi di azzeramento e di impostazione.

Figura u101.30. Realizzazione del flip-flop JK, partendo da un flip-flop D, munito di ingressi di azzeramento e di impostazione. Questa realizzazione è da preferire rispetto a quella della figura successiva.



La figura successiva mostra una realizzazione diversa del flip-flop JK, tratta dal libro *Digital Computer Electronics* di Malvino e Brown, nel capitolo dedicato ai flip-flop. Tuttavia, attraverso l'uso di un simulatore, si determina che se gli ingressi ***J*** e ***K*** vengono attivati e disattivati simultaneamente, prima che ci sia la variazione negativa del clock, al momento della variazione negativa del clock si ottiene ugualmente lo scambio dei valori nelle uscite, mentre ciò non dovrebbe avvenire secondo lo schema di funzionamento previsto per un flip-flop JK.

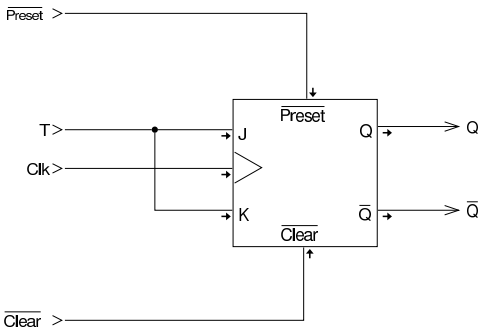
Figura u101.31. Realizzazione del flip-flop JK, partendo da un flip-flop SR in cascata: in tal caso, l'impulso di clock ha effetto in corrispondenza del margine negativo (*negative edge triggered*). Bisogna però fare attenzione al fatto che l'attivazione e successiva disattivazione simultanea dei valori degli ingressi, prima della variazione negativa del clock, provoca lo scambio dei valori delle uscite, come se gli ingressi fossero ancora attivi in quel momento.



È evidente che disponendo di un flip-flop JK è possibile ottenere un

flip-flop T, semplicemente unendo gli ingressi ***J*** e ***K*** che diventano così l'ingresso ***T***.

Figura u101.32. Adattamento di un flip-flop JK per ottenere un flip-flop T.



Registri

Registri semplici 795
Registri a scorrimento 797
Contatori asincroni con flip-flop T 798
Contatori sincroni con flip-flop T 799
Contatori sincroni con flip-flop D 799
Contatori sincroni con caricamento parallelo 800

I ***registri*** sono delle batterie di flip-flop (di norma si tratta di flip-flop D) con le quali è possibile memorizzare valori binari o costruire dei contatori di vario tipo. Di norma i registri si realizzano con flip-flop sincroni che recepiscono il valore in ingresso al momento della variazione dell'impulso di clock (*edge triggered*).

Negli esempi delle sezioni successive si utilizzano flip-flop D, secondo la realizzazione classica, oppure flip-flop derivati da questo tipo. Pertanto, si tratta di flip-flop a margine positivo (pilotati dalla variazione positiva dell'impulso di clock).

Figura u102.1. Flip-flop D secondo la realizzazione classica.

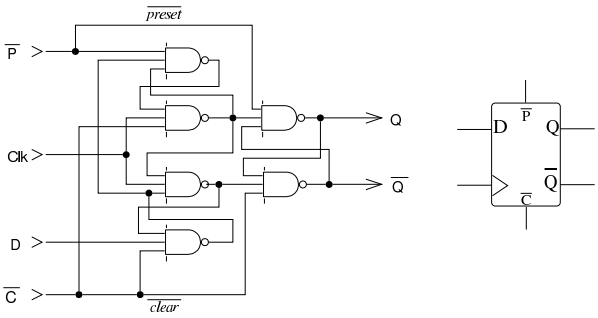
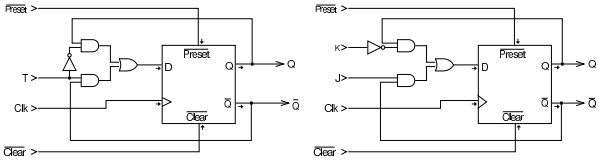


Figura u102.2. flip-flop T e JK, partendo dal flip-flop D.

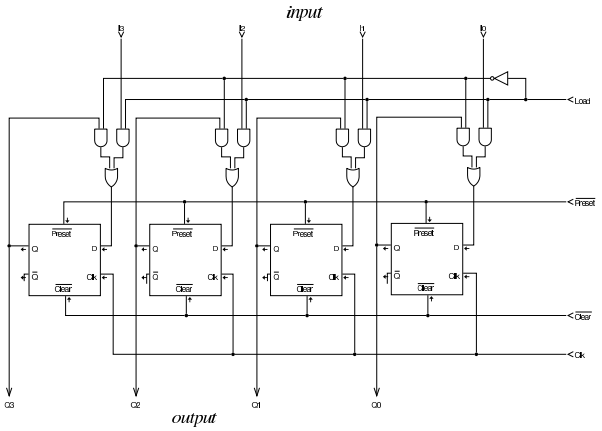


Nelle figure che appaiono nelle sezioni successive, i flip-flop possono avere l'ingresso di clock indicato con la sigla ***Clk*** (senza il triangolo che fa riferimento al margine), tuttavia si tratta sempre di flip-flop pilotati dalla variazione positiva dell'ingresso di clock.

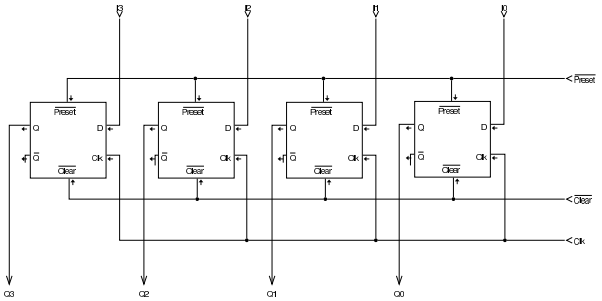
Registri semplici

Il registro più semplice è quello che è in grado di raccogliere un valore composto da più bit e di conservarlo fino a quando non vengono abilitati nuovamente gli ingressi. Nella figura successiva, il valore contenuto nel registro può essere letto dalle uscite da Q_0 a Q_3 e può essere immesso attraverso gli ingressi da I_0 a I_3. Il valore proveniente dagli ingressi da I_0 a I_3, viene raccolto solo quando è attivo l'ingresso ***Load***, altrimenti i flip-flop, a ogni impulso di clock, ricopiano nel proprio ingresso lo stesso valore che mostrano in uscita.

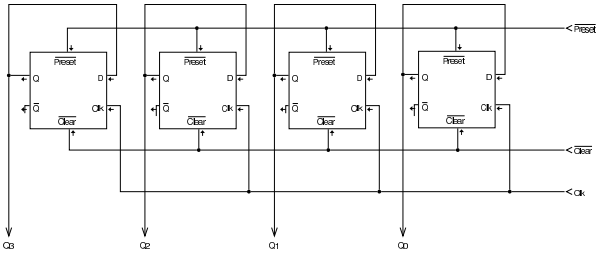
Figura u102.3. Registro a 4 bit: quando l'ingresso ***Load*** è attivo e si presenta una variazione positiva del clock, il registro accumula il valore proveniente dagli ingressi (da I_0 a I_3), altrimenti mantiene il valore accumulato precedentemente.



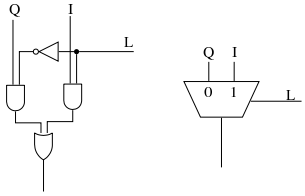
Se si analizza lo schema della figura appena apparsa, si vede che, quando il registro deve accettare il valore in ingresso, è come se il circuito si riducesse all'immagine successiva:



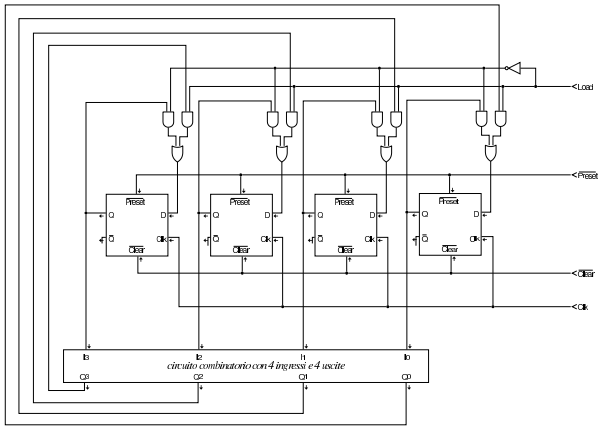
Quando invece si tratta di mantenere il valore memorizzato, è come se il circuito si riducesse allo schema della figura successiva:



Deve essere chiaro che gli ingressi ***D*** di questo tipo di registro sono controllati da un multiplatore, il quale, a ogni impulso di clock, sceglie se recepire un valore nuovo o se rinnovare quello già memorizzato in precedenza.



Una caratteristica importante di un registro comune è quella di potersi aggiornare a partire dal dato che esso stesso contiene, nell'istante in cui viene recepita la variazione dell'ingresso di clock che consente tale aggiornamento. Si osservi l'esempio della figura successiva:

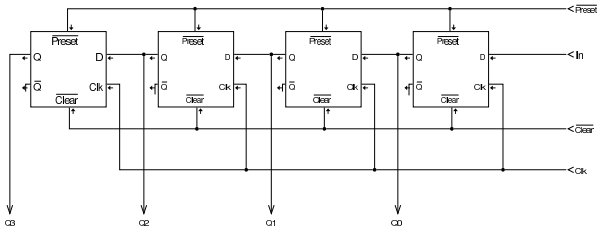


All'uscita del registro, si trova un circuito combinatorio non meglio precisato, il quale trasforma quanto proviene dal registro, fornendo poi il valore trasformato in ingresso al registro stesso: quando l'ingresso ***Load*** è attivo e si manifesta la variazione positiva del segnale di clock, il registro recepisce il nuovo valore trasformato. In pratica, si ottiene quello che in programmazione si rappresenta solitamente come ***x=f(x)***, ovvero l'aggiornamento di una variabile con il risultato di un'espressione che la contiene.

Registri a scorrimento

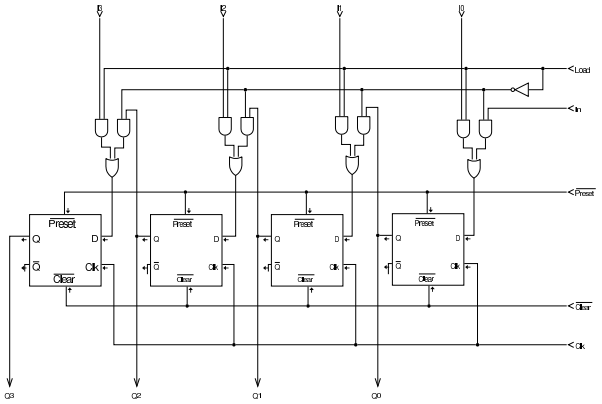
La figura successiva mostra un registro che recepisce un solo valore logico, nell'ingresso ***In***, quando si presenta la variazione positiva del segnale di clock. In quel momento, il primo flip-flop a destra aggiorna il proprio valore con il dato ottenuto dall'ingresso ***In***, mentre il secondo flip-flop si aggiorna ottenendo il valore che il primo flip-flop aveva precedentemente, e così di seguito fino all'ultimo.

Figura u102.8. Registro a scorrimento a 4 bit: quando si verifica una variazione positiva del clock, il valore in ingresso viene accumulato dal primo flip-flop a destra, mentre il secondo recepisce il valore precedente del primo, continuando fino all'ultimo. Video: *http://www.youtube.com/watch?v=I_ncbpQnCZ4* .



Un registro a scorrimento, come quello della figura precedente, può essere esteso in modo da consentire il caricamento di un valore. In tal caso si aggiunge un ingresso (***Load***), con il quale si seleziona la funzione svolta dal registro nel momento della variazione (positiva) del segnale di clock: caricamento o scorrimento.

Figura u102.9. Registro a scorrimento a 4 bit con caricamento: quando l'ingresso ***Load*** è attivo, il registro recepisce il valore dagli ingressi da I_0 a I_3, mentre diversamente fa scorrere il valore contenuto nei flip-flop, acquisendo quanto contenuto nell'ingresso ***In*** nel primo flip-flop. L'acquisizione o lo scorrimento avvengono in corrispondenza di un margine positivo del clock.



Contatori asincroni con flip-flop T

I contatori sono registri che incrementano o decrementano il proprio valore binario a ogni impulso di clock. Sono asincroni quei contatori i cui flip-flop non sono controllati tutti dallo stesso segnale di clock. Il contatore più semplice è costituito da flip-flop T, in quanto questo tipo di flip-flop, quando l'ingresso ***T*** è attivo, inverte il valore delle uscite ogni volta che riceve la variazione del segnale di clock che serve a farlo scattare. Le figure mostrano l'uso dei flip-flop T in cascata, nel senso che l'ingresso di clock del successivo è pilotato dall'uscita del precedente: per questo si tratta di contatori asincroni.

Figura u102.10. Contatore asincrono crescente a 4 bit: il flip-flop successivo è pilotato dall'uscita ***Q'*** del flip-flop precedente. Video: *http://www.youtube.com/watch?v=-mfL6B0w9UI* .

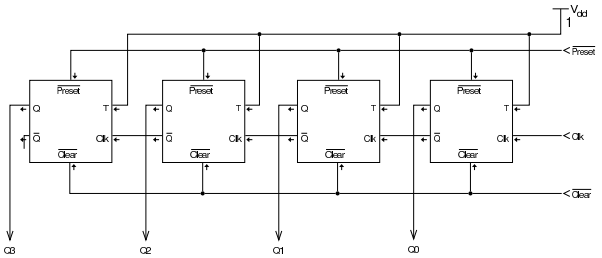
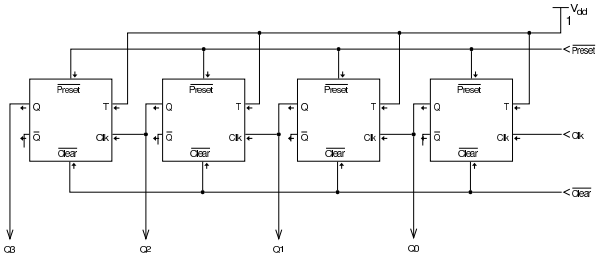


Figura u102.11. Contatore asincrono decrescente a 4 bit: il flip-flop successivo è pilotato dall'uscita ***Q*** del flip-flop precedente. Video: *http://www.youtube.com/watch?v=fvhina5QEnI* .

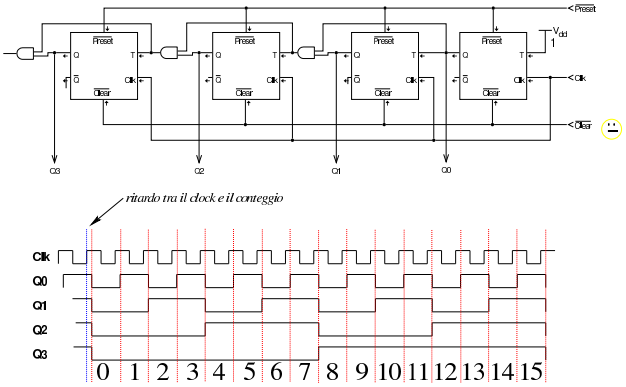


Il difetto dei contatori asincroni sta nel fatto che le oscillazioni delle uscite successive sono leggermente in ritardo rispetto a quelle delle uscite antecedenti. Questo problema si accentua al crescere del rango del registro, perché i ritardi di accumulano a ogni passaggio, da un flip-flop al successivo.

Contatori sincroni con flip-flop T

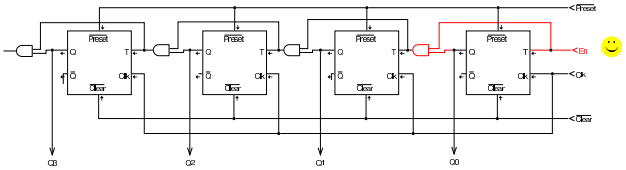
I contatori sincroni possono essere costruiti con flip-flop T, ma in modo differente da quanto descritto nella sezione precedente; tuttavia, quando si modificano questi contatori per consentire l'acquisizione di un valore binario di partenza, è necessaria una costruzione basata su flip-flop D.

Figura u102.12. Contatore sincrono crescente a 4 bit: le uscite sono sincrone tra loro, ma inevitabilmente sono leggermente in ritardo rispetto all'impulso di clock. Video: *http://www.youtube.com/watch?v=Nh5CzRopNdY* .



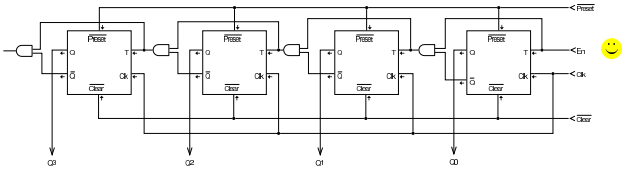
Il contatore sincrono richiede che l'ingresso ***T*** del primo flip-flop sia sempre attivo. Tuttavia, se si controlla tale ingresso è possibile sospendere il processo di conteggio, senza azzerare il valore raggiunto. Pertanto, si può aggiungere un ingresso di abilitazione, da collegare all'ingresso ***T*** del primo flip-flop, proprio per controllare il procedere del conteggio.

Figura u102.13. Contatore sincrono crescente a 4 bit con controllo di abilitazione (ingresso ***En***).



La realizzazione di un contatore sincrono decrescente procede in maniera analoga, invertendo le uscite nella sequenza di flip-flop T.

Figura u102.14. Contatore sincrono decrescente a 4 bit con controllo di abilitazione.



Contatori sincroni con flip-flop D

In generale, è più conveniente realizzare dei contatori sincroni attraverso dei flip-flop D, dato che possono poi essere modificati facilmente per consentire il caricamento parallelo di un valore. Nelle figure di questa sezione si mostrano, per ora, solo contatori equivalenti a quelli già apparsi nella sezione precedente.

Figura u102.15. Contatore sincrono crescente a 4 bit con controllo di abilitazione, basato su flip-flop D.

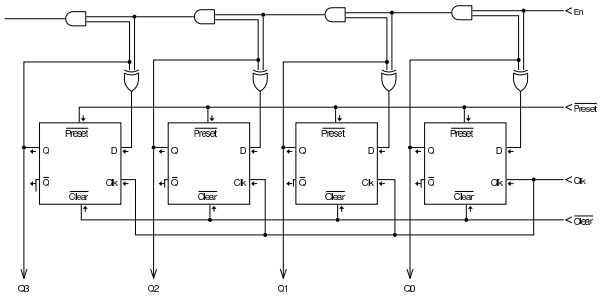
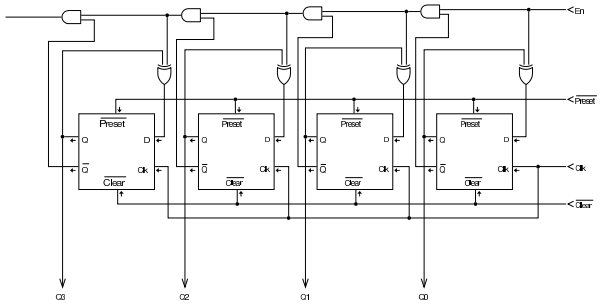


Figura u102.16. Contatore sincrono decrescente a 4 bit con controllo di abilitazione, basato su flip-flop D.



Contatori sincroni con caricamento parallelo

Il contatore è completo solo quando consente anche il caricamento parallelo. Nelle figure di questa sezione si vedono contatori basati su flip-flop D che consentono il caricamento parallelo.

Figura u102.17. Contatore sincrono crescente a 4 bit con caricamento parallelo: l'ingresso ***Load*** fa sì che il valore contenuto negli ingressi da I_0 a I_3 venga caricato nei flip-flop; poi, se l'ingresso ***En*** non è attivo, il valore viene mantenuto tale e quali, altrimenti viene incrementato a ogni impulso di clock.

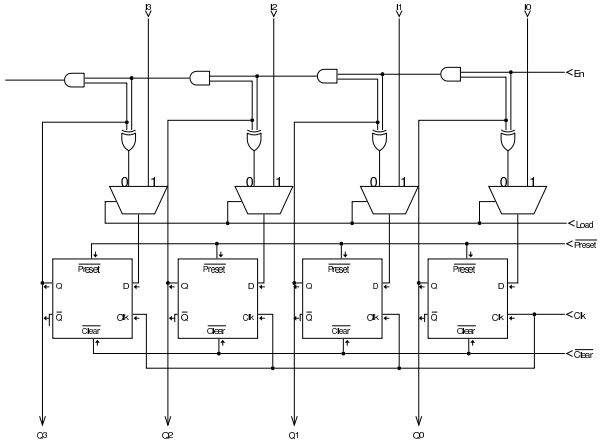


Figura u102.18. Contatore sincrono decrescente a 4 bit con caricamento parallelo: l'ingresso ***Load*** fa sì che il valore contenuto negli ingressi da I_0 a I_3 venga caricato nei flip-flop; poi, se l'ingresso ***En*** non è attivo, il valore viene mantenuto tale e quali, altrimenti viene decrementato a ogni impulso di clock.

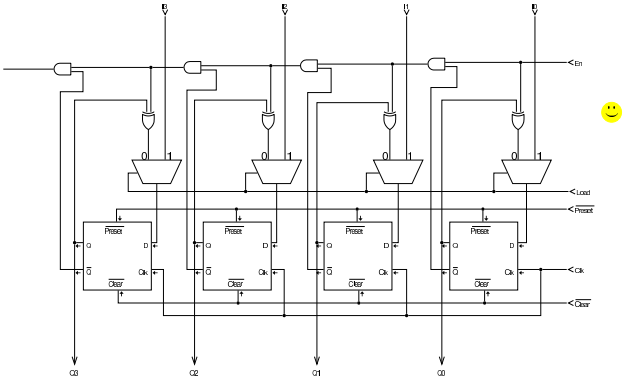
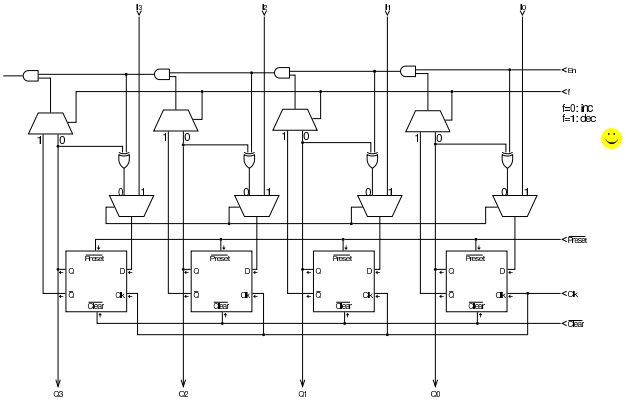


Figura u102.19. Contatore sincrono crescente o decrescente a 4 bit con caricamento parallelo: l'ingresso ***f*** consente di selezionare il funzionamento in qualità di contatore crescente o decrescente.



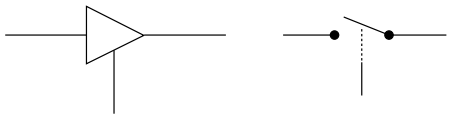
Bus e unità di controllo

Bus con il buffer a tre stati 803
Unità di controllo del bus 804
Microcodice 806

Registri e circuiti combinatori vengono spesso raggruppati condividendo un certo insieme di connessioni (fili). Questo insieme di connessioni costituisce quello che è noto come ***bus***. Un bus nel quale diversi componenti hanno facoltà di scrivere e di leggere è solitamente un «bus dati», perché serve allo scambio di informazioni tra questi componenti; tuttavia, va osservato che solo un componente alla volta può scrivere, mentre non ci sono limitazioni per la lettura concorrenziale.

Bus con il buffer a tre stati

La realizzazione di un bus dati può avvenire con l'ausilio di un multiplatore o di un «buffer a tre stati» (*tri-state buffer*), ovvero un componente che funziona come porta non invertente, ma con un ingresso addizionale, con il quale è possibile abilitare il funzionamento in qualità di porta non invertente (*buffer*), oppure si può isolare completamente l'uscita. Nel disegno seguente si vede a confronto il simbolo del buffer a tre stati con quello che farebbe un interruttore tradizionale:



Il buffer a tre stati, oppure il multiplatore, servono quindi per controllare la scrittura nel bus; tra le due opzioni, la scelta del buffer a tre stati è quella più economica e più semplice, perché il multiplatore comporta l'uso di molte porte logiche e, di conseguenza, introduce un ritardo di propagazione maggiore.

Nelle figure successive si vede l'adattamento di un registro semplice, con buffer a tre stati, per il collegamento a un bus; poi si vede come si possono collegare registri o altri componenti così predisposti, distinguendo tra un bus dati e un bus di controllo.

Figura u103.2. Registro semplice adattato per connettersi con un bus dati.

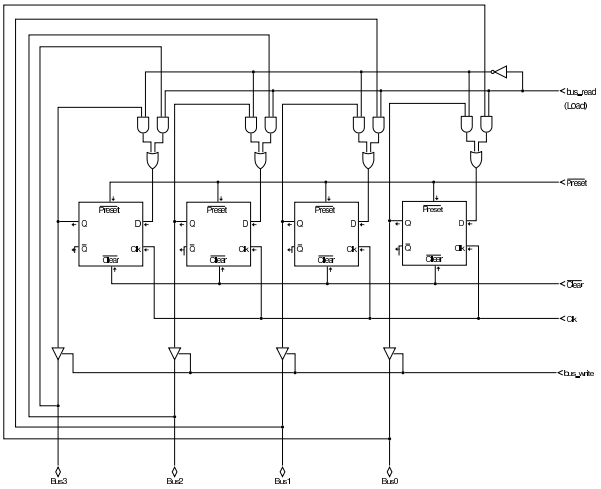


Figura u103.3. Schemi più semplici dell'adattamento di un registro. L'ingresso ***bus_read***, o ***br***, richiede al registro il caricamento del valore che si può leggere dal bus, mentre l'ingresso ***bus_write***, o ***bw***, concede al registro di immettere i propri dati nel bus.

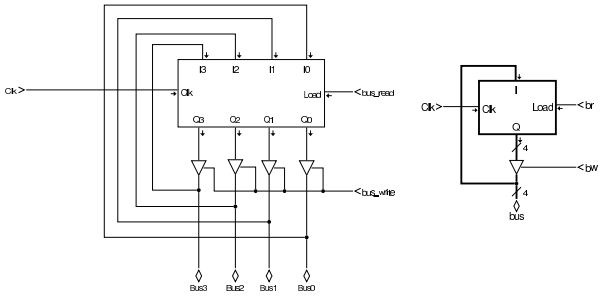
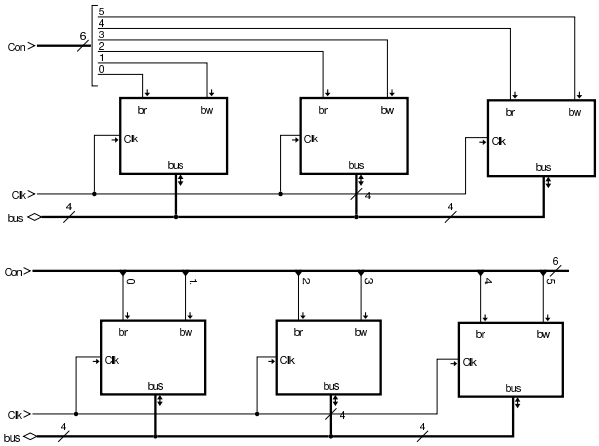


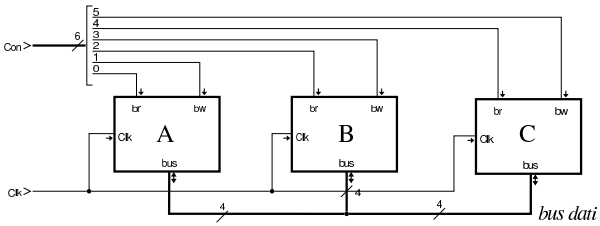
Figura u103.4. Collegamento di più componenti in un bus: gli ingressi che abilitano la lettura dal bus o la scrittura nel bus, assieme ad altri ingressi di controllo eventuali, si collegano a un bus di controllo secondario. Nel secondo disegno si vede un modo alternativo di rappresentare il collegamento al bus di controllo, prelevando un filo alla volta.



Unità di controllo del bus

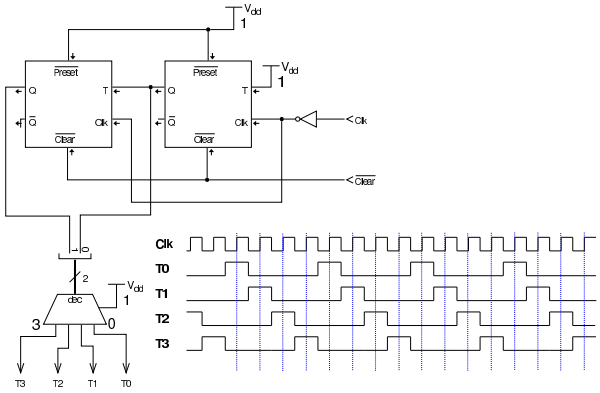
Per dirigere correttamente l'utilizzo di un bus dati, è necessario un sistema di controllo, attraverso il quale scandire le fasi di ogni procedimento che si intende applicare. Per comprendere il meccanismo si riprende lo schema già apparso in cui tre componenti sono connessi su un bus dati e sono controllati da un bus di controllo; su questi componenti si ipotizzano soltanto operazioni di trasferimento delle informazioni.

Figura u103.5. Esempio di riferimento con tre componenti che condividono un bus dati.



Per scandire le operazioni da svolgere, l'unità di controllo ha bisogno di un contatore. Nella realizzazione circuitale di un'unità di controllo si fa riferimento normalmente a un contatore a scorrimento, il quale può essere realizzato come si vede nella figura successiva. Ciò che è importante e che si evidenzia nel tracciato che appare nella figura, è che le uscite risultino attive a cavallo dell'impulso di clock usato per pilotare i componenti del bus. Per ottenere questo risultato, l'impulso di clock che viene usato dal contatore viene invertito.

Figura u103.6. Contatore a scorrimento, usato per l'unità di controllo del bus di esempio: il segnale di clock viene invertito, in modo tale che le uscite T_n siano attive a cavallo del margine positivo che viene usato dai componenti del bus dati. Video: *http://www.youtube.com/watch?v=vXco4E4SNTo* .



L'unità di controllo, realizzata attraverso circuiti logici, deve mettere assieme un decodificatore della funzione da applicare con le uscite del contatore a scorrimento. Per esempio, seguendo lo schema della figura successiva, si vede che la funzione f_1, con la quale si vuole copiare il valore del registro ***A*** nel registro ***B*** e poi dal registro ***B*** nel registro ***C***, il decodificatore attiva la seconda linea a partire dall'alto (etichettata con f_1); quindi, attraverso il collegamento di porte AND, si fa in modo di attivare le uscite ***Aw*** e ***Br*** nel momento in cui il contatore a scorrimento ha l'uscita T_0 attiva; quindi, analogamente, si fa in modo di attivare le uscite ***Bw*** e ***Cr*** nel momento in cui il contatore a scorrimento ha l'uscita T_1 attiva. ***Aw***, ***Br***, e ***Bw*** e ***Cr***, attivate secondo la scansione descritta, vanno ad attivare gli ingressi di lettura-scrittura dei registri rispettivi, consentendo il trasferimento di dati previsto. La realizzazione dell'unità di controllo della figura successiva è estremamente semplificata e il tempo T_2 non viene nemmeno utilizzato, comportando quindi un'attesa inutile in quel momento; tuttavia, il tempo T_3 serve invece per bloccare il segnale di clock nell'unità di controllo, la quale richiede di essere azzerata per poter recepire un nuovo comando dall'ingresso ***f*** .

Figura u103.7. Unità di controllo: sono previste solo quattro funzioni molto semplici, per le quali bastano solo due tempi; pertanto il tempo T_2 rimane inutilizzato e il tempo T_3 serve a bloccare l'unità di controllo fino a quando l'ingresso ***Run*** viene azzerato e poi riattivato. Video: *http://www.youtube.com/watch?v=r-DZgjy-ya0* .

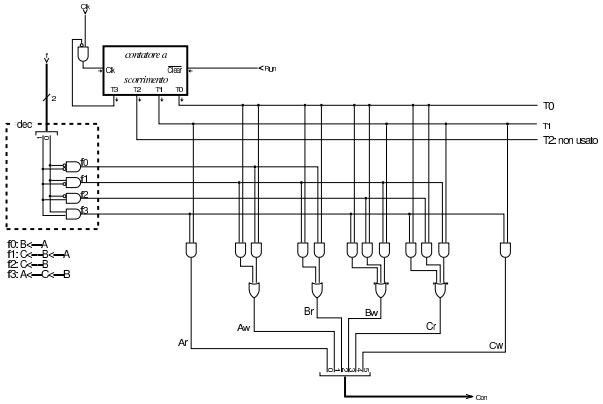
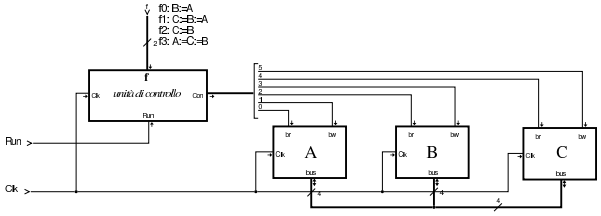


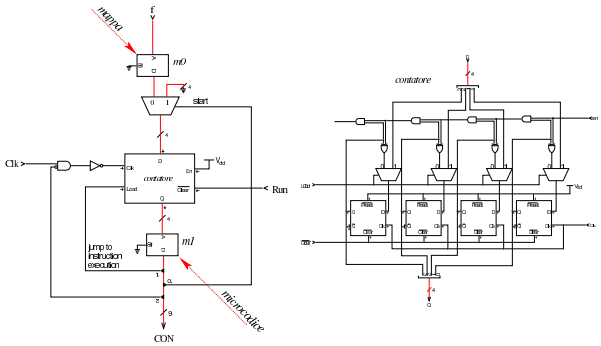
Figura u103.8. Unità di controllo connessa ai componenti che deve pilotare. Video: *http://www.youtube.com/watch?v=kpET2kEcUIo* .



Microcodice

La realizzazione di un'unità di controllo di un bus dati, sotto forma di circuito logico, può risultare in un lavoro estremamente complesso. Per questa ragione, si utilizza solitamente una memoria ROM (in sola lettura), suddivisa in due parti: ***mappa*** e ***microcodice***. Le due parti della memoria ROM vanno viste come due tabelle: la prima traduce la funzione desiderata in un indirizzo che punta alla seconda, dove inizia il codice che descrive i vari passaggi da eseguire. Si osservi la figura successiva, dove a sinistra c'è lo schema dell'unità di controllo e a destra c'è lo schema del contatore utilizzato all'interno della stessa.

Figura u103.9. Unità di controllo realizzata attraverso memorie ROM: a sinistra lo schema a blocchi dell'unità, a destra la scomposizione del contatore a quattro bit (un semplice contatore incrementante, con possibilità di caricare un valore, basato su flip-flop D). Video: *http://www.youtube.com/watch?v=mBGhNP3Uujs* .



Le due memorie ROM sono denominate, rispettivamente, ***m0*** ed ***m1***. Nella figura successiva si vede il contenuto delle due memorie, rappresentato in forma tabellare, mettendo in corrispondenza l'indirizzo in ingresso della memoria (ingresso ***A***) e il contenuto che viene rappresentato nell'uscita (***D***). La memoria ***m0*** di questi esempi utilizza solo due bit per gli indirizzi, i quali corrispondono alla funzione richiesta all'unità di controllo; in uscita, la stessa memoria, produce un valore a quattro bit che rappresenta, a sua volta, un indirizzo per la memoria ***m1***.

Figura u103.10. Contenuto e collegamento tra le memorie ROM ***m0*** e ***m1*** che compongono il microcodice.

m_0

	indirizzo	contenuto	
f_0	0	0001	(1)
f_1	1	0011	(3)
f_2	2	0110	(6)
f_3	3	1000	(8)

m_1

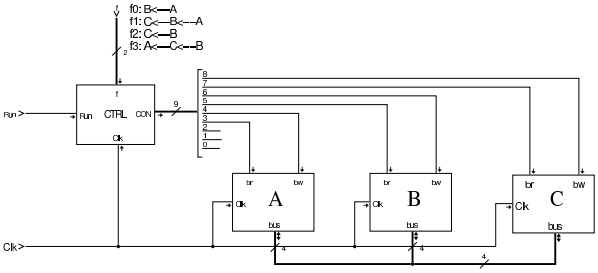
indirizzo	contenuto	
0	000000010	*load counter*
1	000110000	*B <-- A*
2	000000111	*load zero, stop clock*
3	000110000	*B <-- A*
4	011000000	*C <-- B*
5	000000111	*load zero, stop clock*
6	011000000	*C <-- B*
7	000000111	*load zero, stop clock*
8	011000000	*C <-- B*
9	100001000	*A <-- C*
10	000000111	*load zero, stop clock*

L'uscita della memoria ***m0*** viene inviata a un contatore; l'uscita del contatore serve ad accedere alla memoria ***m1***; l'uscita della memoria ***m1*** è ciò che serve per controllare il bus, con l'aggiunta di qualche linea per controllare la stessa unità. Per comprendere cosa succede, vengono scanditi i vari passaggi nell'elenco successivo, ipotizzando che sia richiesta la funzione f_1.

1. All'avvio l'ingresso ***Run*** è a zero, cosa che comporta l'azzeramento del contatore. Così facendo, dalla memoria ***m1*** viene selezionato l'indirizzo zero, dal quale si ottiene il valore 000000010_2. Il bit attivo, di questo valore, viene usato per richiedere al contatore il caricamento dell'indirizzo che riceve in ingresso, proveniente dalla memoria ***m0***, la quale lo produce in base alla funzione richiesta all'unità di controllo.
2. Attivando l'ingresso ***Run***, appena si presenta una **variazione negativa** del valore di ***Clk*** (l'ingresso ***Clk*** è invertito per questo) il contatore carica l'indirizzo proveniente dalla memoria ***m0*** (0011_2) e lo riproduce nell'ingresso della memoria ***m1***, dalla quale si ottiene il valore 000110000_2, corrispondenti alla richiesta di trasferire il contenuto di ***A*** in ***B***.
3. Si presenta un'altra variazione negativa del valore di ***Clk*** e il contatore viene incrementato di un'unità, selezionando da ***m1*** l'indirizzo 0100_2, con il quale la memoria ***m1*** produce il valore 011000000_2, corrispondenti alla richiesta di trasferire il contenuto di ***B*** in ***C***.
4. Si presenta un'altra variazione negativa del valore di ***Clk*** e il contatore viene incrementato di un'unità, selezionando da ***m1*** l'indirizzo 0101_2, con il quale la memoria ***m1*** produce il valore 000000111_2, corrispondenti alla richiesta di: fornire in ingresso al contatore il valore zero, caricare il contatore (con il valore zero), bloccare l'ingresso ***Clk***.

A questo punto il ciclo di esecuzione di una funzione è terminato e, per eseguirne un'altra, dopo aver fornito il valore corrispondente alla nuova funzione occorre disattivare e riattivare l'ingresso ***Run***.

Figura u103.11. Collegamento dell'unità di controllo ai componenti del bus dati: le prime tre linee del bus di controllo sono utilizzate dall'unità stessa e non servono a pilotare i moduli del bus dati.



Ciò che si scrive nelle memorie ROM che compongono l'unità di controllo può essere prodotto con strumenti che ne consentono una rappresentazione simbolica. Tkgate dispone di un compilatore che produce i file-immagine del microcodice, della mappa che consente di raggiungere le istruzioni nel microcodice attraverso la specificazione dei codici operativi, ed eventualmente anche il macrocodice, a partire da un sorgente unico. Il listato successivo mostra un sorgente compatibile con l'esempio di questa sezione.

Listato u103.12. Microcodice e codice operativo scritto per Tkgate 2.

```
map       bank[1:0]  m0;
microcode bank[8:0]  m1;
//-------------------------------------------------------
field ctrl_start[0];    // parte dall'indirizzo 0
field ctrl_load[1];     // carica l'indirizzo nel contatore.
field stop[2];          // stop clock.
field a_br[3];          // A <-- bus
field a_bw[4];          // A --> bus
```

```
field b_br[5];          // B <-- bus
field b_bw[6];          // B --> bus
field c_br[7];          // C <-- bus
field c_bw[8];          // C --> bus
//-------------------------------------------------------
op f0 {
  map f0: 0;
  +0[7:0]=0;
};
op f1 {
  map f1: 1;
  +0[7:0]=1;
};
op f2 {
  map f2: 2;
  +0[7:0]=2;
};
op f3 {
  map f3: 3;
  +0[7:0]=3;
};
//-------------------------------------------------------
begin microcode @ 0
load:
    ctrl_load;                  // CNT <-- TBL[f]

f0:
    b_br a_bw;                  // B <-- A
    ctrl_start ctrl_load stop;  // CNT <-- 0

f1:
    b_br a_bw;                  // B <-- A
    c_br b_bw;                  // C <-- B
    ctrl_start ctrl_load stop;  // CNT <-- 0

f2:
    c_br b_bw;                  // C <-- B
    ctrl_start ctrl_load stop;  // CNT <-- 0

f3:
    c_br b_bw;                  // C <-- B
    a_br c_bw;                  // A <-- C
    ctrl_start ctrl_load stop;  // CNT <-- 0

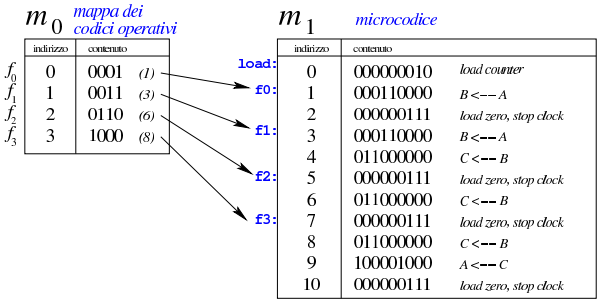
end
```

Il contenuto della memoria che rappresenta la «mappa» è organizzato in «codici operativi». Per la precisione, il codice operativo è l'indirizzo della mappa in cui si trova, a sua volta, l'indirizzo della memoria contenente il codice operativo, da cui inizia l'esecuzione di una certa funzione. Per esempio, il codice operativo della funzione f_2 è 2, come si vede nel listato di Tkgate:

```
op f2 {
  map f2: 2;
  +0[7:0]=2;
};
```

Viene riproposta la figura che mette a confronto le due memorie, ***m0*** e ***m1***, con l'aggiunta di altri dettagli, a completamento dell'argomento.

Figura u103.14. Mappa e microcodice.



Tkgate

File PostScript ed EPS 809
Rappresentazione di multiplatori, demultiplatori e codificatori 809
Clock 811

Tkgate[1] (*http://www.tkgate.org*) è un programma per il disegno assistito (CAD) di circuiti logici, con la possibilità di utilizzare un simulatore, per il quale è possibile costruire anche dei moduli in grado di interfacciarsi con il sistema operativo ospitante.

La versione più recente di Tkgate è incompleta; tuttavia è migliore rispetto a quella ritenuta stabile. In questa sezione si annotano solo alcuni accorgimenti per poter utilizzare proficuamente la versione 2.0-b10 (beta) di Tkgate, nonostante i piccoli difetti che presenta. Va comunque osservato che la versione 2.0-b***n*** di Tkgate va compilata personalmente, dato che difficilmente si può trovare un pacchetto pronto per la propria distribuzione; per la compilazione occorre installare nel sistema i sorgenti delle librerie Tcl/Tk 8.5.

File PostScript ed EPS

I disegni realizzati attraverso Tkgate possono essere «stampati» producendo file PostScript o EPS (a seconda delle opzioni selezionate in fase di stampa). I file PostScript o EPS generati da Tkgate 2.0-b10 hanno un difetto importante che impedisce loro di essere gestiti da Ghostscript. Si osservi l'estratto seguente di codice PostScript/EPS:

```
/Courier-Latin1 /Courier findfont defLatin1
/Courier-Bold-Latin1 /Courier-Bold findfont defLatin1
/Courier-Italic-Latin1 /Courier-Italic findfont defLatin1
/Courier-BoldItalic-Latin1 /Courier-BoldItalic findfont defLatin1
/Helvetica-Latin1 /Helvetica findfont defLatin1
/Helvetica-Bold-Latin1 /Helvetica-Bold findfont defLatin1
/Helvetica-Oblique-Latin1 /Helvetica-Oblique findfont defLatin1
```

Le righe evidenziate in nero provocano un errore irreversibile per Ghostscript, ma è sufficiente commentarle senza per questo impoverire il risultato tipografico del file:

```
/Courier-Latin1 /Courier findfont defLatin1
/Courier-Bold-Latin1 /Courier-Bold findfont defLatin1
%/Courier-Italic-Latin1 /Courier-Italic findfont defLatin1
%/Courier-BoldItalic-Latin1 /Courier-BoldItalic findfont defLatin1
/Helvetica-Latin1 /Helvetica findfont defLatin1
/Helvetica-Bold-Latin1 /Helvetica-Bold findfont defLatin1
/Helvetica-Oblique-Latin1 /Helvetica-Oblique findfont defLatin1
```

Per correggere questi file con l'aiuto di uno script, si può usare SED (sezione 23.5) con un comando come quello seguente:

```
$ cat file.ps↩
↪    | sed "s/^\/Courier-Italic/%\/Courier-Italic/g"↩
↪    | sed "s/^\/Courier-BoldItalic/%\/Courier-BoldItalic/g"↩
↪     > fix.ps [Invio]
```

Rappresentazione di multiplatori, demultiplatori e codificatori

I componenti del tipo multiplatori e simili, hanno dei terminali numerati, per sapere in che ordine sono disposti. Tkgate consente di invertire l'ordine, rispetto a quello predefinito (da sinistra a destra); tuttavia, nella rappresentazione stampata (file PostScript o EPS), questi componenti vengono mostrati sempre come se l'ordine dei terminali fosse quello predefinito. Va osservato che questo problema esiste comunque anche nella versione stabile di Tkgate.

Per risolvere il problema della stampa, conviene annotare con dei commenti l'ordine dei terminali del multiplatore e degli altri componenti simili, poi è opportuno sopprimere il codice PostScript responsabile di questo errore.

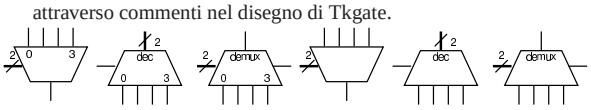
Nel file PostScript o EPS, ogni componente ha una sua «funzione», dichiarata nel preambolo. L'estratto seguente mostra le funzioni di multiplatore e componenti analoghi, dove è stato commentato

il codice responsabile della rappresentazione errata dell'ordine dei terminali:

```
/mux {
    dup /mrot exch def
    startgate
    8 rfont
    -29.5 15.5 moveto
    29.5 15.5 lineto
    16.5 -12.5 lineto
    -16.5 -12.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
    mrot -90 eq mrot -180 eq or {
%       3 -1 roll 1 sub 50 string cvs exch (0) exch      % d1 (n) (0) dn
%       -29 add 7 rCT                                    % d1
%       exch -29 add 7 rCT
    } {
%       3 -1 roll 1 sub 50 string cvs exch       % d1 (n) dn
%       -29 add 7 rCT                            % d1
%       (0) exch -29 add 7 rCT
    } ifelse
    grestore
} def
…
/demux {
    startgate
    8 rfont
    (demux) 0 5 rCT
    -16.5 12.5 moveto
    16.5 12.5 lineto
    29.5 -15.5 lineto
    -29.5 -15.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
%    3 -1 roll 1 sub 50 string cvs exch % d1 (n) dn
%    -29 add -12 rCT                              % d1
%    (0) exch -29 add -12 rCT
    grestore
} def
…
/decoder {
    startgate
    8 rfont
    (dec) 0 5 rCT
    -16.5 12.5 moveto
    16.5 12.5 lineto
    29.5 -15.5 lineto
    -29.5 -15.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
%    3 -1 roll 1 sub 50 string cvs exch % d1 (n) dn
%    -29 add -12 rCT                              % d1
%    (0) exch -29 add -12 rCT
    grestore
} def
```

Il codice PostScript/EPS non può essere corretto diversamente, perché queste funzioni non ricevono alcun parametro che informi loro dell'ordine in cui vanno rappresentati i terminali.

Figura u104.4. Multiplatore, decodificatore e demultiplatore: a sinistra nella loro rappresentazione originale che, però, potrebbe essere errata, a destra togliendo i numeri d'ordine dei terminali. Se si tolgono i numeri d'ordine, vanno aggiunte delle indicazioni attraverso commenti nel disegno di Tkgate.



Clock

Tra i moduli già pronti che accompagnano Tkgate, ne esistono due che servono a generare un segnale di *clock*, ovvero un'onda quadra, strutturata in qualche modo. Si tratta precisamente della libreria di moduli denominata '**`timer`**', all'interno della quale sono disponibili i moduli '**`ONESHOT`**' e '**`OSCILLATOR`**'. Questi moduli sono scritti usando codice Verilog di Tkgate.

I due moduli prevedono un parametro, denominato ***HZ***, con il quale si presume di inserire una frequenza espressa in Hz, ma non è così: osservando il codice, si può notare che si tratta invece della

durata che dovrebbe avere l'onda o l'impulso, espressa in millisecondi (ms). Tuttavia, pur contando questo fatto, la frequenza che si ottiene nel simulatore è molto differente, perché durante la simulazione non c'è alcuna connessione con l'orologio del sistema operativo ospitante, al quale si rimettono invece i moduli '**ONESHOT**' e '**OSCILLATOR**'. In pratica, questi due moduli vanno usati con il valore della «frequenza» predefinita, osservando nel simulatore se la frequenza effettiva può andare bene per i propri fini. Va anche osservato che è inutile tentare di modificare il codice di questi moduli, in modo da produrre effettivamente la frequenza desiderata, proprio perché il simulatore non riuscirebbe a farne buon uso.

Per poter disporre di un generatore di frequenza che, in qualche modo, possa essere controllato in relazione al metro usato nella simulazione, occorre costruire qualcosa che si basi sui ritardi di propagazione. Per prima cosa serve un generatore di un impulso; precisamente serve qualcosa che parta producendo un valore azzerato, per poi passare ad attivarsi dopo un certo tempo, mantenendosi attivo per tutto il tempo successivo. Per produrre questo componente virtuale, occorre predisporre un modulo scritto direttamente secondo il linguaggio Verilog di Tkgate:

```
module one_up #(.W(1000)) (Z);
  output Z;
  reg Z;

  initial
    begin
      Z = 1'b0;
      $tkg$wait(W);
      Z = 1'b1;
    end

endmodule
```

Si tratta di un modulo privo di entrate e avente una sola uscita: all'avvio l'uscita si presenta a zero e ci rimane per il tempo specificato dal parametro ***W***, espresso in millisecondi, quindi passa a uno e ci rimane fino alla fine. Il tempo di pausa iniziale serve a consentire ai componenti di inizializzarsi; il tempo predefinito di 1000 ms, corrispondente a 1 s, viene visto durante la simulazione come un tempo molto più breve, ma generalmente sufficiente: nel caso potrebbe essere aumentato.

Disponendo del modulo di inizializzazione, si può realizzare un circuito che produca un'oscillazione sfruttando il ritardo di propagazione dei componenti; poi, questa oscillazione può essere suddivisa convenientemente in base alle esigenze. Nel capitolo xxvi si mostra un oscillatore realizzato in questo modo.

[1] **Tkgate** GNU GPL

Riferimenti

- *Tkgate*, *http://www.tkgate.org*
- Mario Italiani, Giuseppe Serazzi, *Elementi di informatica*, ETAS libri, 1973, ISBN 8845303632
- Albert Paul Malvino, Jerard A. Brown, *Digital Computer Electronics*, Glencoe/Mcgraw-Hill *http://www.amazon.it/Digital-Computer-Electronics-Albert-Malvino/dp/0028005945*
- Stephen Brown, Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, Mcgraw-Hill *http://www.amazon.com/Fundamentals-Mcgraw-Hill-Electrical-Computer-Engineering/dp/0072823151* ; *Flip-Flops, Registers, Counters, and a Simple Processor*, *http://highered.mcgraw-hill.com/sites/dl/free/0072823151/56549/vra23151_ch07.pdf*
- Luigi Zeni, *Elettronica dei sistemi digitali*, *http://www.dii.unina2.it/Utenti/lzeni/Elettronica%20de20i%20Si20stemi%20Di20gitali/ESD-CircuitiCombinatori.pdf* , *http://www.dii.unina2.it/Utenti/lzeni/Elettronica%20de20i%20Si20stemi%20Di20gitali/ESD-CircuitiSequenziali.pdf*
- CL GATE aspire, *Combinational & Sequential Circuits*, *http://media.careerlauncher.com.s3.amazonaws.com/gate/material/2.pdf*
- *CSC 4210/6210 – Computer Architecture*, *http://www.cs.gsu.edu/~cscagb/csc4210/*
- Olivier Carton, *Circuits et architecture des ordinateurs*, *http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/archi.pdf*
- Tony R. Kuphaldt, *Lessons In Electric Circuits*, *http://www.faqs.org/docs/electric/*
- *Building a Digital Computer*, *http://artematrix.org/Projects/TTL.processor/processor.design.htm*

Versione A: caricamento ed esecuzione del codice 817
Versione B: indice della memoria 831
Istruzioni «load» 833
Istruzioni «store» 833
Versione C: registri generici 837
Versione D: ALU 841
Istruzione «not» 846
Istruzione «and» 846
Istruzione «or» 847
Istruzione «xor» 848
Istruzioni «lshl» e «lshr» 849
Istruzioni «ashl» e «ashr» 850
Istruzioni «rotl» e «rotr» 851
Istruzione «add» 852
Istruzione «sub» 853
Versione E: indicatori 855
Istruzione «rotcl» e «rotcr» 857
Istruzione «add_carry» 858
Istruzione «sub_borrow» 860
Versione F: condizioni 863
Versione G: pila 869
Istruzioni «push» e «pop» 873
Istruzioni «call» e «return» 873
Versione H: I/O 875
Generalizzazione della comunicazione con i dispositivi 875
Realizzazione dei dispositivi di I/O 876
Aspetto e funzionamento esteriore delle interfacce sincrone 878
Interfaccia sincrona della tastiera 879
Interfaccia sincrona dello schermo 880
Il bus della CPU con i dispositivi di I/O 881
Istruzione «out» 883
Istruzione «in» 884
Versione I: ottimizzazione 885
Registri uniformi 885
RAM 886
Modulo «SEL» 888
ALU 888
Terminale 889
Unità di controllo 891
Memorie, campi, argomenti e codici operativi 893
Microcodice 899
Macrocodice: chiamata di una routine 907
Macrocodice: inserimento da tastiera e visualizzazione sullo schermo 907
Versione J: ottimizzazione bis 909
Versione K: 16 bit «little-endian» 911
Registri a 16 bit 911
Modulo «BUS» 913
Modulo «ALU» 913
Modulo «SEL» 916
Modulo «RAM» 917
Modulo «IRQ» 918

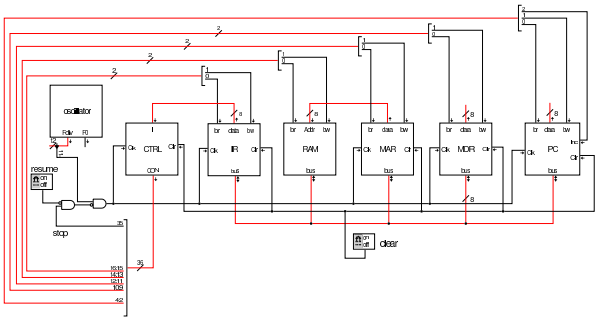
Modulo «IVT» 920
Modulo «CTRL» 921
Codici operativi 923
Microcodice 934
Gestione delle interruzioni 942
Orologio: modulo «RTC» 944
Modulo «TTY» 944
Modulo «HDD» 945
Macrocodice: esempio di uso del terminale con le interruzioni 947
Riferimenti 949

Viene qui introdotto lo sviluppo di una CPU dimostrativa, aggiungendo progressivamente componenti e funzioni, fino ad arrivare a un elaboratore molto semplice. Inizialmente si tratta solo di una CPU con registri a 8 bit, inclusi quelli relativi all'indirizzamento della memoria RAM, la quale è limitata così a un massimo di 256 byte.

Versione A: caricamento ed esecuzione del codice

Nella sua prima versione, la CPU si compone soltanto di registri utili ad accedere alla memoria per leggere il codice operativo da eseguire, come di vede nella figura successiva.

Figura u106.1. Il bus della CPU nella sua prima fase realizzativa.



Il modulo più semplice che si può analizzare è l'oscillatore che serve a produrre il segnale di clock. Si tratta di un oscillatore costruito con una serie di porte logiche invertenti, per creare un ritardo di propagazione sufficiente a produrre un'oscillazione a una frequenza gestibile. Per attivare l'oscillazione si richiede un impulso iniziale che, dopo una breve pausa a zero, si attiva stabilmente. La figura successiva mostra l'oscillatore e l'impulso di avvio necessario per l'attivazione. È importante osservare che la serie di porte invertenti deve essere in numero dispari, come se si trattasse di una sola porta invertente, ma con un lungo ritardo di propagazione. Il risultato viene poi passato a un divisore di frequenza, composto in questo caso da una catena di flip-flop T, sincroni, in modo da non sfasare l'oscillazione a ogni divisione; in uscita si hanno tante linee raggruppate assieme, ognuna delle quali permette di prelevare un'oscillazione a una frequenza differente. Il divisore di frequenza è inizializzato dallo stesso impulso iniziale, il quale parte da uno stato a zero. Nel caso degli esempi viene usata una frequenza molto bassa, corrispondente all'ultimo stadio di divisione.

Figura u106.2. Oscillatore utilizzato per il segnale di clock.

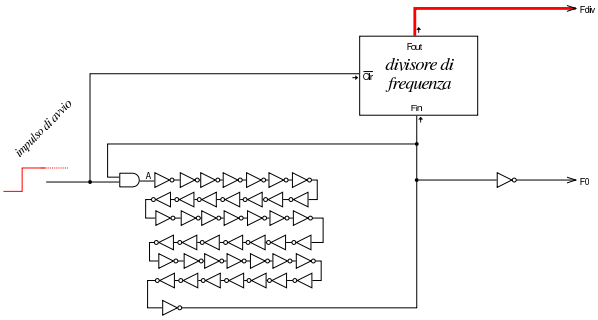
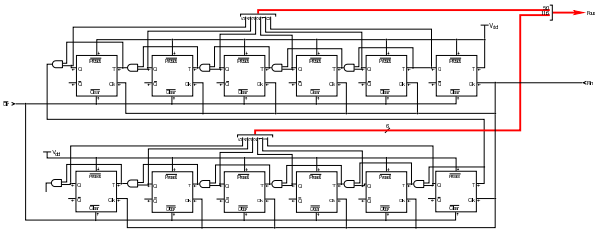


Figura u106.3. Divisore utilizzato nel modulo dell'oscillatore.



L'impulso iniziale viene prodotto da un componente sintetizzato attraverso del codice Verilog, in quanto diversamente servirebbero

componenti elettronici non logici e la loro trattazione esula dallo scopo di questo studio.

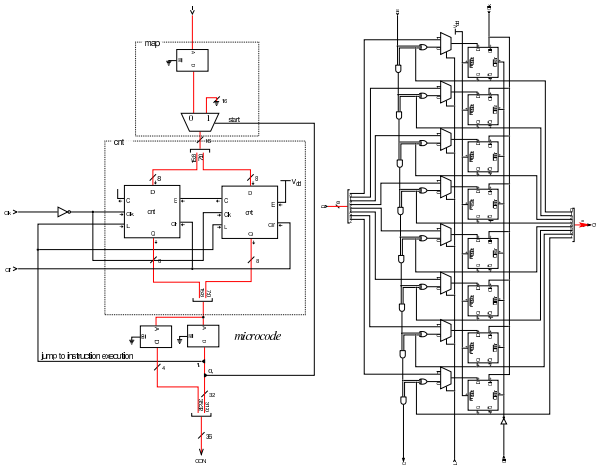
Figura u106.4. Codice Verilog per Tkgate, relativo al modulo di innesco dell'oscillazione: l'uscita è inizialmente a zero e dopo un breve istante passa a uno, rimanendo così stabilmente. Il tempo di attesa iniziale è configurabile attraverso il parametro ***W***.

```
module one_up #(.W(1000)) (Z);
  output Z;
  reg Z;
  initial
    begin
      Z = 1'b0;
      $tkg$wait(W);
      Z = 1'b1;
    end
endmodule
```

L'unità di controllo, contenuta nel modulo **`CTRL`**, è molto simile a quella descritta nella sezione u0.3, con la differenza che l'ingresso è individuato dalla variabile ***I*** a 8 bit (la lettera «I» sta per «istruzione») e che l'uscita ha un rango molto maggiore, costringendo a utilizzare due unità di memoria in parallelo. Il contatore che serve a scandire le istruzioni nel blocco finale di memoria è complessivamente a 16 bit, ma per convenienza, ne sono stati usati due da 8 in cascata.

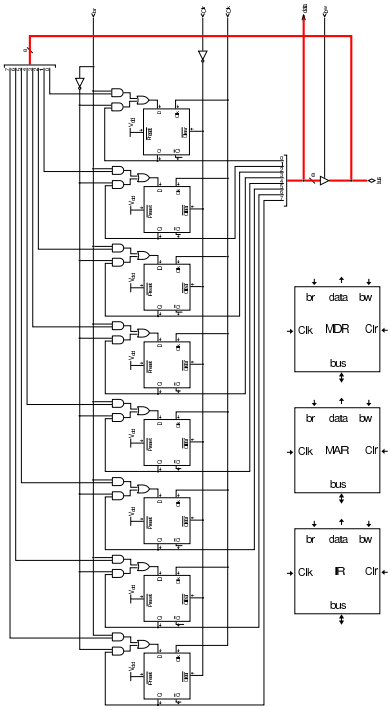
L'ingresso ***I*** dell'unità di controllo è alimentato dal contenuto del registro ***IR*** (*instruction register*).

Figura u106.5. Unità di controllo, evidenziando a destra la struttura del modulo **`cnt`** che rappresenta un contatore, basato su flip-flop D, estensibile per ottenere ranghi maggiori.



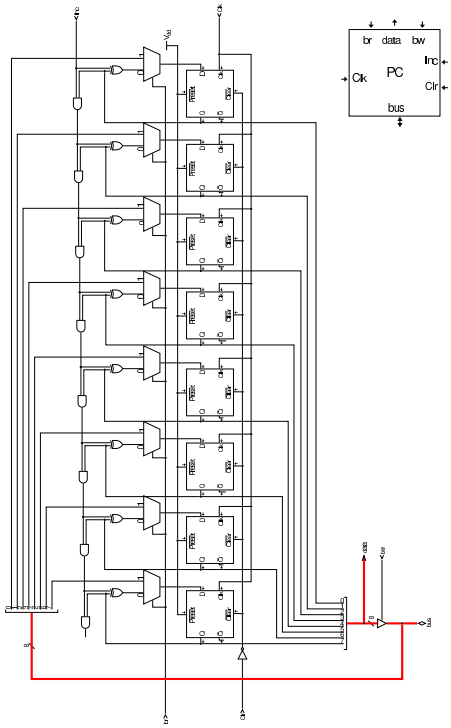
I moduli **`IR`**, **`MAR`** e **`MDR`**, sono registri semplici, costruiti con flip-flop D, connessi al bus attraverso dei buffer a tre stati, dai quali è possibile prelevare copia del valore memorizzato da un'uscita supplementare, denominata ***data***. Il registro ***IR*** (*instruction register*), a cui si è già accennato, ha lo scopo di conservare il codice operativo che l'unità di controllo deve eseguire; il registro ***MAR*** (*memory address register*) ha lo scopo di conservare l'indirizzo di memoria a cui si vuole accedere; il registro ***MDR*** (*memory data register*) serve ad accumulare quanto viene letto dalla memoria per qualche motivo o ciò che vi deve essere scritto.

Figura u106.6. Registri ***IR***, ***MAR*** e ***MDR***.



Il modulo **`PC`** è un registro simile agli altri, con la differenza che può incrementare il valore che contiene quando è attivo l'ingresso ***Inc***. Il registro ***PC*** (*program counter*) ha lo scopo di contenere l'indirizzo di memoria del codice successivo da eseguire.

Figura u106.7. Registro contatore ***PC***.

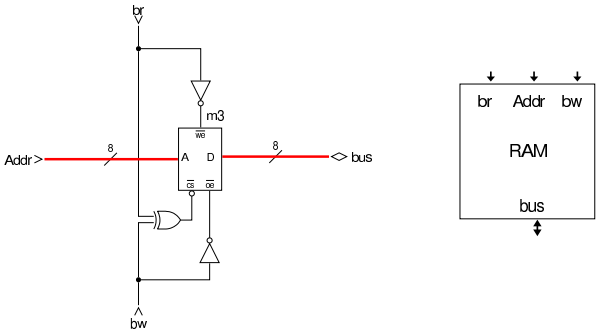


Il modulo **`RAM`** è sostanzialmente differente dagli altri, in quanto racchiude la memoria RAM usata dalla CPU. A tale memoria si accede attraverso l'indirizzo fornito tramite l'ingresso ***Addr***, a 8 bit, e anche il contenuto della memoria è organizzato a celle da 8 bit. Il modulo condivide con gli altri gli ingressi di controllo dell'accesso

al bus; tuttavia, quando il modulo riceve l'indirizzo ed è abilitata la lettura dal bus, il valore contenuto in memoria viene aggiornato subito (salvo il ritardo di propagazione), senza attendere l'impulso di clock.

Il modulo **RAM** riceve l'indirizzo dal registro ***MAR*** (*memory address register*), il quale è così dedicato a contenere e conservare l'indirizzo di memoria a cui si vuole accedere.

Figura u106.8. Modulo **RAM**. La rete logica che controlla gli ingressi ***br*** e ***bw***, serve a impedire che si possa mettere in pratica la lettura e scrittura simultanea del bus.



La prima cosa di cui si deve occupare la struttura appena descritta, consiste nel caricamento di un'istruzione, seguito poi dall'esecuzione della stessa: ciò è noto come ***ciclo di caricamento*** (*fetch*). Nella struttura in questione, il registro ***PC*** contiene l'indirizzo dell'istruzione da eseguire: questo valore deve essere trasferito nel registro ***MAR*** e il registro ***PC*** viene incrementato; dalla memoria RAM si ottiene l'istruzione contenuta nell'indirizzo ***MAR*** che viene copiata nel registro ***IR***. Ciò si può rappresentare sinteticamente come segue:

1. ***MAR*** = ***PC***
2. ***PC***++
3. ***IR*** = ***RAM***[***MAR***]

Le figure successive mostrano proprio questi tre passaggi, evidenziando i valori degli ingressi ***br***, ***bw*** e ***Inc***, attraverso dei LED che diventano rossi nel momento dell'attivazione della linea a cui sono connessi. Le figure mostrano sempre solo il momento in cui il segnale di clock diventa attivo.

Figura u106.9. Prima fase: si richiede al registro ***PC*** di inviare il suo valore al bus e al registro ***MAR*** di leggerlo. Si attua in pratica l'operazione ***MAR***=***PC***.

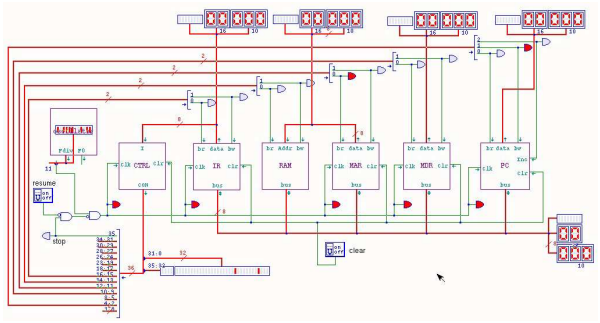


Figura u106.10. Seconda fase: si richiede al registro ***PC*** di incrementarsi di una unità. Si attua in pratica l'operazione ***PC***++.

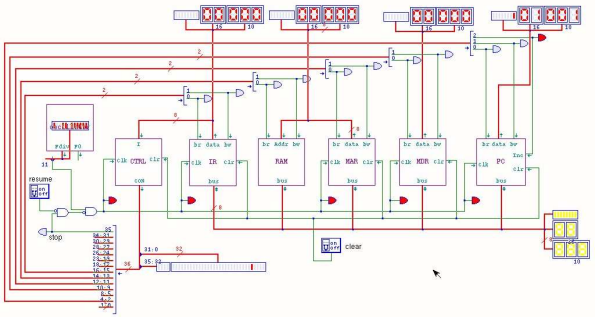
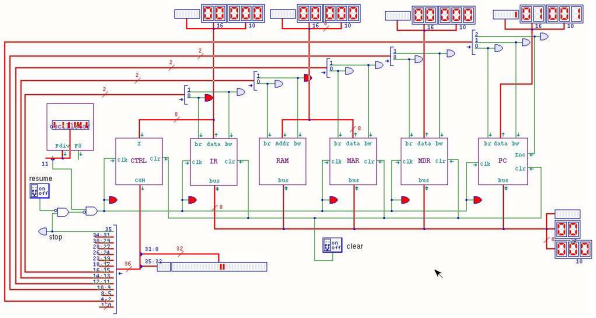


Figura u106.11. Terza fase: si richiede alla RAM di inviare il valore corrispondente all'indirizzo recepito dal registro ***MDR*** al bus e al registro ***IR*** di accumulare questo valore. Si attua in pratica l'operazione ***IR***=***RAM***[***MAR***].



All'interno dell'unità di controllo (il modulo **CTRL**) il tempo è scandito allo stesso modo, a parte il fatto che i contatori ***cnt*** sono pilotati da un segnale di clock invertito, per anticipare l'attivazione delle linee di controllo rispetto all'impulso relativo alla gestione del bus dati. Inizialmente i contatori dell'unità di controllo si trovano a essere azzerati e per questo vanno a ricercare nella memoria sottostante la prima microistruzione, corrispondente alla richiesta di eseguire l'operazione ***MAR***=***PC***. Successivamente il complesso dei due contatori ***cnt*** viene incrementato e ciò fa passare alla seconda microistruzione, corrispondente alla richiesta di incremento del registro ***PC***. Nel terzo istante si ha un incremento ulteriore, facendo emergere la microistruzione ***IR***=***RAM***[***MAR***].

Figura u106.12. Prima fase: i contatori dell'unità di controllo sono azzerati e la microistruzione iniziale corrisponde a ***MAR***=***PC***.

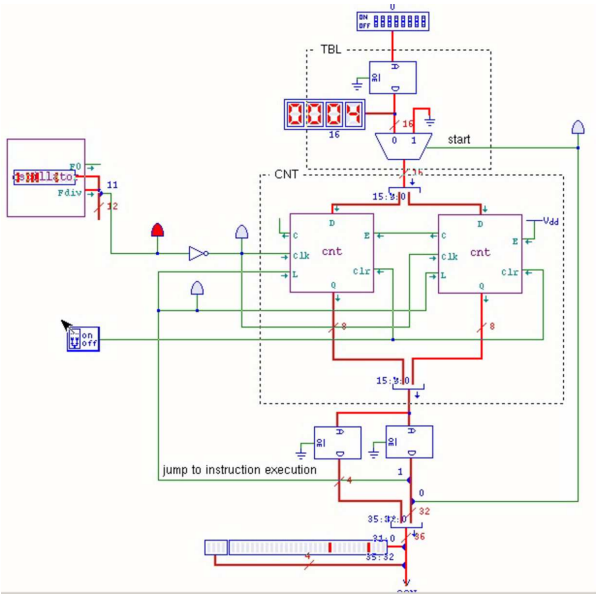


Figura u106.13. Seconda fase: il complesso dei contatori è stato incrementato e la microistruzione prodotta corrisponde a ***PC***++.

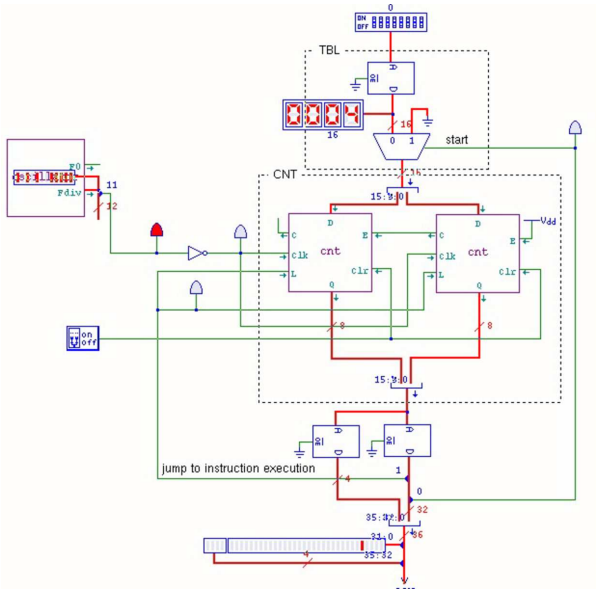
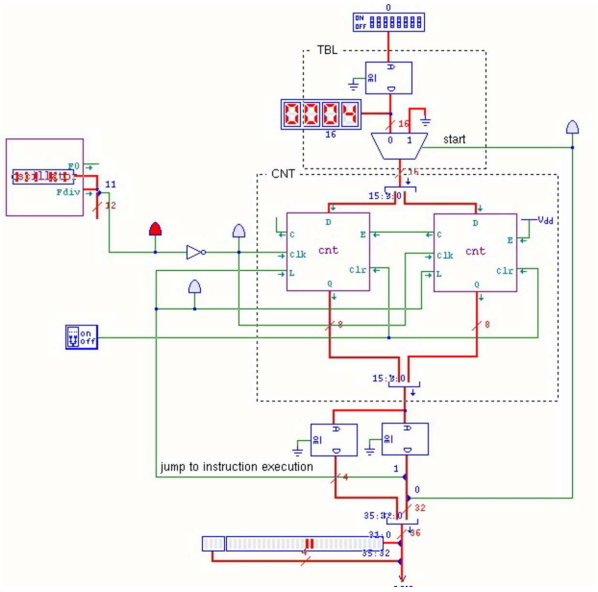


Figura u106.14. Terza fase: il complesso dei contatori è stato incrementato e la microistruzione prodotta corrisponde a ***IR***=***RAM***[***MAR***].



A questo punto, l'unità di controllo dispone dell'istruzione da eseguire nell'ingresso ***I*** ed è pronta per recepirla. Per farlo, la microistruzione successiva richiede al contatore interno di accettare il valore in ingresso. Questo valore corrisponde al contenuto della memoria ***m0***, la quale tratta l'istruzione in ingresso come indirizzo, dal quale produce a sua volta l'indirizzo del microcodice successivo a cui saltare. Negli esempi delle figure, l'istruzione in questione corrisponde al codice operativo 00000000_2, ovvero all'istruzione nulla (`not_operate`).

Figura u106.15. Quarta fase: i contatori dell'unità di controllo sono caricati con il valore proveniente dalla memoria che traduce l'istruzione in indirizzo del microcodice.

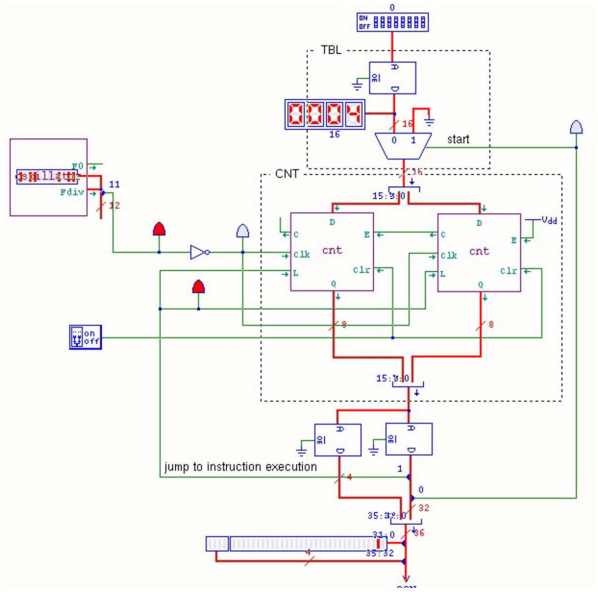
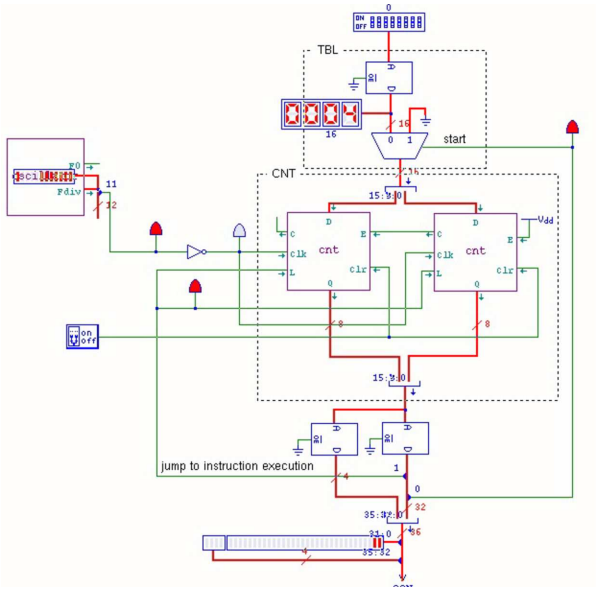


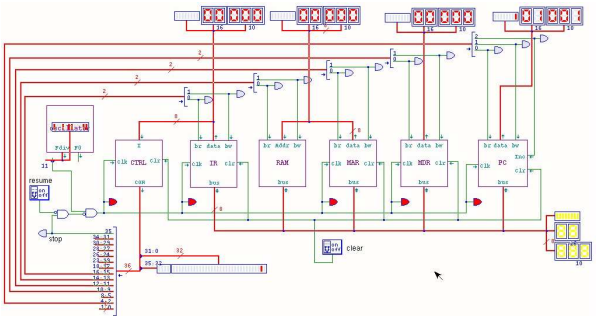
Figura u106.16. Fase conclusiva: i contatori dell'unità di controllo sono stati incrementati e puntano alla microistruzione successiva. Dal momento che l'istruzione originale (**not_operate**) non richiedeva lo svolgimento di alcuna operazione nel bus dati, ci si trova al termine della procedura per tale istruzione, incontrando la microistruzione che richiede al complesso di contatori dell'unità di controllo di azzerarsi. L'azzeramento avviene facendo caricare ai contatori il valore zero, tramite il multiplatore che controlla l'ingresso di tali contatori.



Dopo l'azzeramento dei contatori dell'unità di controllo, si ricomincia dal microcodice iniziale (le prime tre fasi) con il quale si richiede il caricamento di una nuova istruzione.

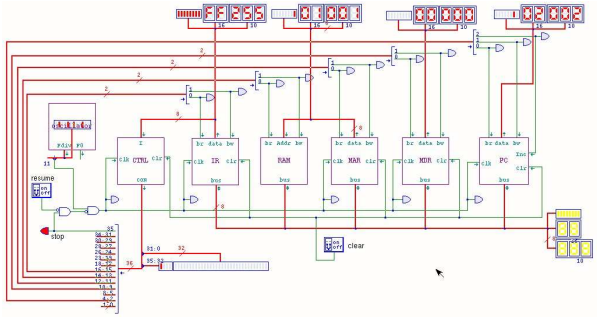
Va osservato che durante la quarta fase (salto al microcodice di esecuzione dell'istruzione richiesta) e durante la fase conclusiva (salto al microcodice iniziale che attua il ciclo di caricamento), nel bus dati non succede nulla.

Figura u106.17. Durante la fase di salto al microcodice di esecuzione dell'istruzione richiesta e durante il salto al microcodice del ciclo di caricamento, nel bus dati non succede nulla.



Per fermare il funzionamento del circuito descritto, esiste l'istruzione **stop** (11111111_2), con la quale viene fermato il segnale di clock. La figura successiva mostra questa situazione.

Figura u106.18. La situazione in cui si trova il bus dati quando viene eseguita l'istruzione **stop**: la linea di controllo CON_{35} si attiva e va a bloccare il segnale di clock. Per far riprendere l'esecuzione da quel punto, superando lo stop, occorrerebbe intervenire nell'interruttore situato vicino al LED che risulta attivo.



Dovrebbero essere disponibili due video, nei quali si dimostra l'esecuzione di due sole istruzioni (macroistruzioni):

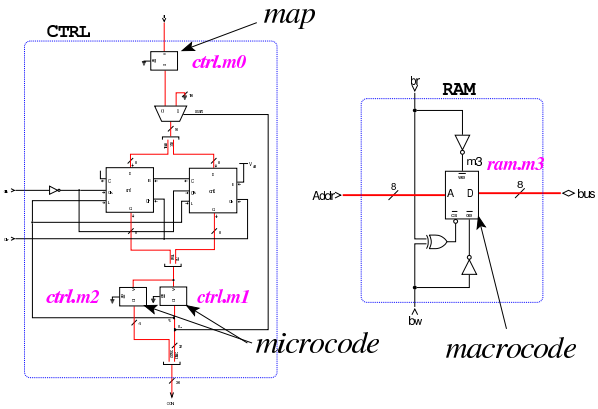
1. **not_operate**
2. **stop**

Il primo video *http://www.youtube.com/watch?v=Z8bTO8WjYYc* mostra ciò che accade nel bus dati; il secondo, invece, mostra l'interno dell'unità di controllo *http://www.youtube.com/watch?v=pPxCQz7IFbM* .

Per descrivere il contenuto delle memorie, incluso quello della memoria RAM, viene usato un file sorgente scritto secondo la sintassi adatta a '**gmac**' di Tkgate 2. Le prime direttive descrivono i banchi di memoria, i quali sono organizzati così: ***ctrl.m0*** corrisponde alla prima memoria in alto dell'unità di controllo; ***ctrl.m1*** e ***ctrl.m2*** sono le due memorie che contengono il microcodice e che si trovano in basso nello schema dell'unità di controllo; ***ram.m3*** è invece la memoria contenuta nel modulo **RAM** del bus dati e ospita il macrocodice che inizialmente si limita solo a **not_operate** e **stop**.

```
map       bank[7:0]   ctrl.m0;
microcode bank[31:0]  ctrl.m1;
microcode bank[35:32] ctrl.m2;
macrocode bank[7:0]   ram.m3;
```

Figura u106.20. Dove si trovano concretamente i banchi di memoria.



Si passa quindi alla descrizione dei campi in cui è suddivisa ogni cella di memoria che rappresenta il microcodice (***ctrl.m1*** e ***ctrl.m2***). Per esempio, il bit meno significativo si chiama ***ctrl_start***, mentre il più significativo si chiama ***stop***. Va osservato che non sono descritti tutti i 36 bit della cella che rappresenta una microistruzione, perché al momento il codice si limita a rappresentare la riduzione della CPU nella sua prima versione.

```
field ctrl_start[0];    // parte dall'indirizzo 0
field ctrl_load[1];     // carica l'indirizzo nel contatore.
field pc_br[2];         // PC <-- bus
field pc_bw[3];         // PC --> bus
field pc_Inc[4];        // PC++
field mdr_br[9];        // MDR <-- bus
field mdr_bw[10];       // MDR --> bus
field mar_br[11];       // MAR <-- bus
field mar_bw[12];       // MAR --> bus
field ram_br[13];       // RAM[mar] <-- bus
field ram_bw[14];       // RAM[mar] --> bus
field ir_br[15];        // IR <-- bus
field ir_bw[16];        // IR --> bus
field stop[35];         // stop clock
```

Vengono poi descritti i tipi di operandi che possono avere le istruzioni (le macroistruzioni). Si prevede di gestire istruzioni senza operandi, oppure con un solo operando di 8 bit. Il significato della sintassi utilizzata per descrivere il tipo ***op_0*** e il tipo ***op_1***, va approfondito, eventualmente, nella documentazione di Tkgate.

```
operands op_0 {
    //
    // [........]
    //
    -  = { };
};
operands op_1 {
    //
    // [........][nnnnnnnn]
    //
    #1 = { +1=#1[7:0]; };
};
```

Si passa poi alla descrizione dei codici operativi; per esempio, si vede che l'istruzione **`not_operate`** corrisponde al codice zero (00000000_2), mentre l'istruzione **`jump`** ha il codice 15 (00001111_2). Va osservato che nel primo caso (**`not_operate`**) non ci sono argomenti, mentre nel secondo si richiede un argomento.

```
op not_operate {
  map not_operate : 0;
  +0[7:0]=0;
  operands op_0;
};
op jump {
  map jump : 15;                    // jump to #nn
  +0[7:0]=15;
  operands op_1;
};
op stop {
  map stop : 255;                   // stop
  +0[7:0]=255;
  operands op_0;
};
```

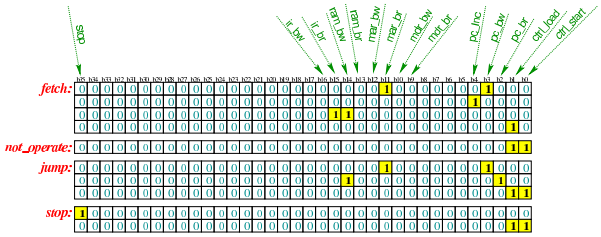
Inizia quindi la definizione del microcodice, il quale viene collocato a partire dall'indirizzo zero della coppia di memorie ***ctrl.m1*** e ***ctrl.m2***. Si può osservare che si inizia proprio dalla descrizione del ciclo di caricamento (*fetch*) che si conclude con il salto alla microistruzione che inizia la procedura che mette in pratica la macroistruzione recepita; inoltre, alla fine della descrizione di ogni macroistruzione (in forma di microcodice), viene richiesto di saltare nuovamente alla prima microistruzione, con la quale si ripete il ciclo di caricamento.

```
begin microcode @ 0
//
fetch:
    mar_br pc_bw;               // MAR = PC
    pc_Inc;                     // PC++
    ir_br ram_bw;               // IR = RAM[MAR]
    ctrl_load;                  // salta alla
                                // microistruzione
                                // corrispondente
//
not_operate:
    ctrl_start ctrl_load;       // salta a «fetch»
//
```



```
jump:
    mar_br pc_bw;               // MAR = PC
    pc_br ram_bw;               // PC <-- RAM[MAR]
    ctrl_start ctrl_load;       // salta a «fetch»
//
stop:
    stop;                       // stop clock.
    // Se il clock fosse
    // riabilitato manualmente:
    ctrl_start ctrl_load;       // salta a «fetch»
//
end
```

Figura u106.25. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).



Infine, inizia il macrocodice, ovvero il codice assemblatore da immettere nella memoria RAM:

```
begin macrocode @ 0
start:
        not_operate
        stop

end
```

Figura u106.27. Macrocodice contenuto nella memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

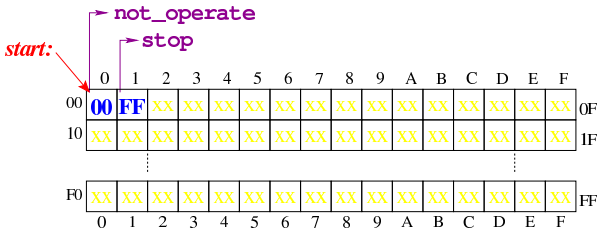


Tabella u106.28. Elenco delle macroistruzioni di questa prima versione della CPU dimostrativa.

Sintassi	Descrizione
`not_operate`	Non fa alcunché, limitandosi a passare all'istruzione successiva.
`jump` ***indirizzo***	Salta all'istruzione che si trova in memoria all'indirizzo specificato.
`stop`	Ferma l'afflusso degli impulsi di clock.

Il file descritto dovrebbe essere disponibile all'indirizzo *allegati/circuiti-logici/scpu-sub-a.gm* . Per compilarlo con '**`gmac`**' di Tkgate 2, si dovrebbe procedere con il comando successivo:

```
$ gmac20 -o scpu-sub-a.mem -m scpu-sub-a.map ↩
↪       scpu-sub-a.gm [Invio]
```

Il file '`scpu-sub-a.mem`' che si ottiene è quello che serve a Tkgate 2 per caricare i contenuti delle memorie previste. Eventualmente, dovrebbe essere disponibile anche il file *allegati/circuiti-logici/scpu-sub-a.v* che contiene la rappresentazione completa di questa prima versione della CPU dimostrativa nel formato di Tkgate 2.

Prima di concludere la descrizione della versione iniziale della CPU dimostrativa, va osservato che esiste una terza istruzione che non è ancora stata usata in un esempio: **`jump`**. Questa si realizza semplicemente con i passaggi seguenti:

1. ***MAR*** = ***PC***

2. ***PC*** = ***RAM***[***MAR***]

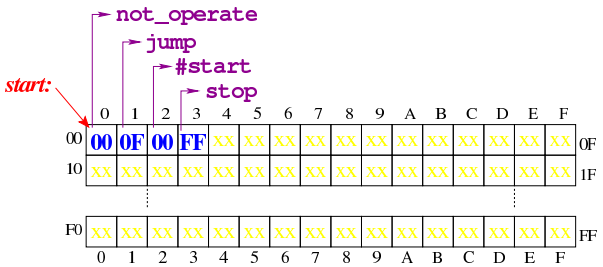
In pratica: nel registro ***MAR*** viene copiato l'indirizzo contenuto nel registro ***PC***, il quale corrisponde all'indirizzo successivo all'istruzione appena letta e in corso di esecuzione (**`jump`**), ma il contenuto della memoria corrispondente a tale indirizzo, viene copiato di nuovo nel registro ***PC*** (senza incrementarlo).

L'istruzione **`jump`** precede un argomento, costituito dall'indirizzo a cui si vuole saltare incondizionatamente; pertanto, tale indirizzo si colloca subito dopo il codice dell'istruzione e viene letto attraverso l'indice del registro ***PC***, come se si trattasse di un'istruzione; poi, però, il contenuto della memoria in corrispondenza di quell'indirizzo, non viene inviato al registro ***IR***, ma viene immesso nuovamente nel registro ***PC***, in maniera tale che la prossima istruzione a essere caricata sia quella a cui si vuole saltare.

A titolo di esempio, il macrocodice (ovvero il codice assemblatore) potrebbe essere modificato come segue:

```
begin macrocode @ 0
start:
        not_operate
        jump #start
        stop
end
```

Figura u106.30. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.



Durante la compilazione, '**`#start`**' viene rimpiazzato dall'indirizzo corrispondente all'etichetta '**`start:`**' che in pratica è semplicemente zero. Questo piccolo programma si limita a non fare nulla (**`not_operate`**) e a ripeterlo indefinitivamente, tanto che l'istruzione **`stop`** non può mai essere eseguita. Le figure successive mostrano ciò che accade dopo l'esecuzione dell'istruzione **`not_operate`** nel bus dati.

Figura u106.31. La situazione in cui si trova il bus dati quando è stata caricata l'istruzione **`jump`** e il registro ***PC***, puntando all'indirizzo che segue l'istruzione **`jump`**, immette il suo valore nel registro ***MAR***.

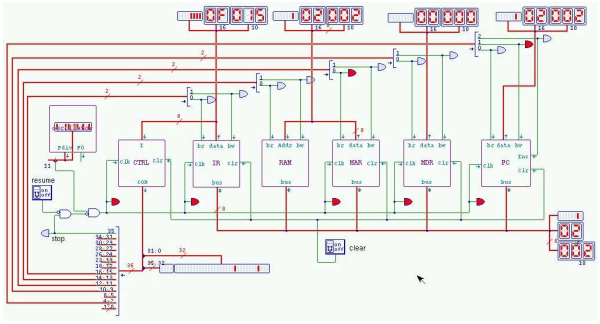
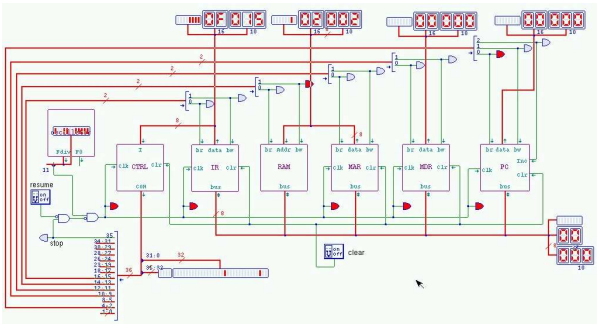


Figura u106.32. Il valore contenuto nella memoria, in corrispondenza dell'indirizzo di salto, viene immesso nel registro ***PC***, facendo in modo che si riparta poi da quella posizione.



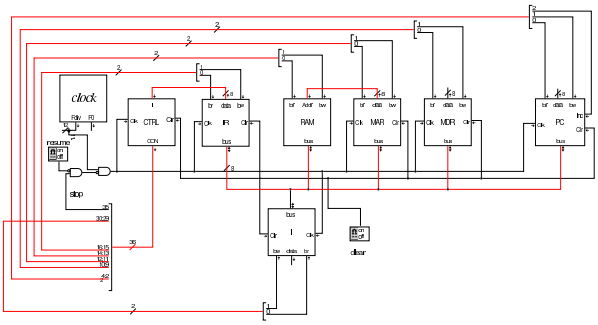
Dovrebbe essere disponibile un video che mostra l'esecuzione del macrocodice descritto: *http://www.youtube.com/watch?v=Z8bTO8WjYYc* .

Versione B: indice della memoria

Istruzioni «load» 833
Istruzioni «store» 833

Nella seconda versione della CPU dimostrativa, viene aggiunto soltanto un registro speciale, denominato ***I***, il cui scopo è quello di contenere un indice della memoria. Nello specifico, serve a poter leggere o scrivere nella memoria RAM, attraverso un indice che possa essere gestito. Il registro ***I*** è realizzato nello stesso modo di ***MDR***, ***MAR*** e ***IR***.

Figura u107.1. Il bus della CPU nella sua seconda fase realizzativa.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti, i quali servono specificatamente a gestire il registro ***I***:

```
field i_br[29];          // I <-- bus
field i_bw[30];          // I --> bus
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op load_imm
{
  map load_imm : 1;                 // load from address #nn
  +0[7:0]=1;
  operands op_1;
};
op load_reg
{
  map load_reg : 2;                 // load from address %I
  +0[7:0]=2;
  operands op_0;
};
op store_imm {
  map store_imm : 3;                // store to address #nn
  +0[7:0]=3;
  operands op_1;
};
op store_reg {
  map store_reg : 4;                // store to address I
  +0[7:0]=4;
  operands op_0;
};
op move_mdr_i {
  map move_mdr_i : 11;              // move MDR to I
  +0[7:0]=11;
  operands op_0;
};
op move_i_mdr {
  map move_i_mdr : 12;              // move I to MDR
  +0[7:0]=12;
  operands op_0;
};
```

```
begin microcode @ 0
…
load_imm:
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    // La memoria non ha un clock,
```

```
    // quindi, non si può passare
    // direttamente a MAR.
    i_br ram_bw;                // I <-- RAM[MAR]
    mar_br i_bw;                // MAR <-- I
    mdr_br ram_bw;              // MDR <-- RAM[MAR]
    ctrl_start ctrl_load;       // CNT <-- 0
//
load_reg:
    mar_br i_bw;                // MAR <-- I
    mdr_br ram_bw;              // MDR <-- RAM[MAR]
    ctrl_start ctrl_load;       // CNT <-- 0
//
store_imm:
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    i_br ram_bw;                // I <-- RAM[MAR]
    mar_br i_bw;                // MAR <-- I
    ram_br mdr_bw;              // RAM[MAR] <-- MDR
    ctrl_start ctrl_load;       // CNT <-- 0
//
store_reg:
    mar_br i_bw;                // MAR <-- I
    ram_br mdr_bw;              // RAM[MAR] <-- MDR
    ctrl_start ctrl_load;       // CNT <-- 0
//
move_mdr_i:
    i_br mdr_bw;                // I <-- MDR
    ctrl_start ctrl_load;       // CNT <-- 0
//
move_i_mdr:
    mdr_br i_bw;                // MDR <-- I
    ctrl_start ctrl_load;       // CNT <-- 0
…
end
```

Figura u107.5. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

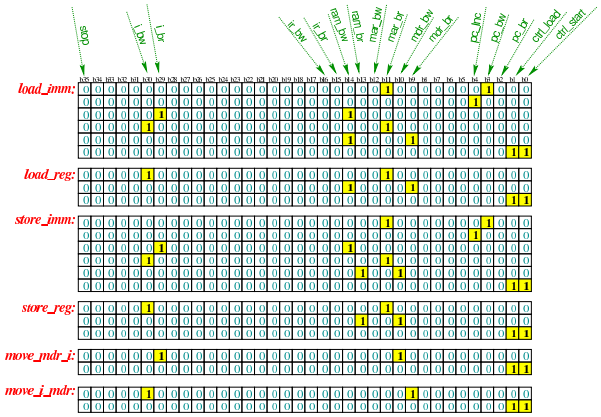


Tabella u107.6. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`load_imm` ***indirizzo***	*load immediate*: carica nel registro ***MDR*** il contenuto della cella di memoria che corrisponde all'indirizzo indicato dall'argomento. Contestualmente, il registro ***I*** viene modificato e alla fine contiene l'indirizzo di memoria in questione.
`load_reg`	*load register*: carica nel registro ***MDR*** il contenuto della cella di memoria che corrisponde all'indirizzo indicato dal valore contenuto nel registro ***I***.
`store_imm` ***indirizzo***	*store immediate*: salva in memoria, all'indirizzo specificato come argomento, il valore contenuto nel registro ***MDR***. Contestualmente, il registro ***I*** viene modificato e alla fine contiene l'indirizzo di memoria in questione.

Sintassi	Descrizione
`store_reg`	*store register*: salva in memoria, all'indirizzo specificato dal registro ***I***, il valore contenuto nel registro ***MDR***.
`move_mdr_i`	Copia il contenuto del registro ***MDR*** nel registro ***I***.
`move_i_mdr`	Copia il contenuto del registro ***I*** nel registro ***MDR***.

Istruzioni «load»

Come primo esempio viene proposto il macrocodice seguente:

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_i
        load_reg
stop:
        stop
data_1:
       .byte 3
end
```

In pratica, viene caricato nel registro ***MDR*** il valore corrispondente all'indirizzo in cui si trova l'etichetta '`data_1:`' (facendo i conti si tratta dell'indirizzo 5); successivamente, il valore di ***MDR*** viene copiato nel registro ***I*** e quindi viene caricato nel registro ***MDR*** quanto contenuto nell'indirizzo di memoria corrispondente al valore di ***I***: dal momento che a quel indirizzo si trova il valore 2, corrispondente al codice operativo dell'istruzione **`load_reg`**, al termine, il registro ***MDR*** contiene tale valore. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile all'indirizzo *allegati/circuiti-logici/scpu-sub-b.gm*

Figura u107.8. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

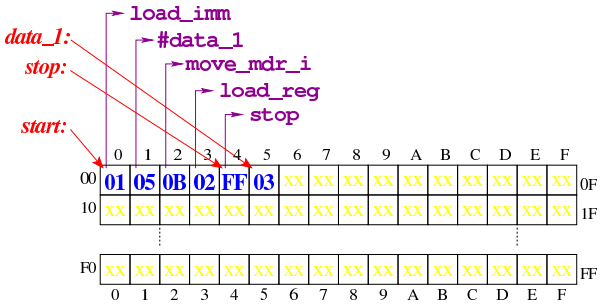
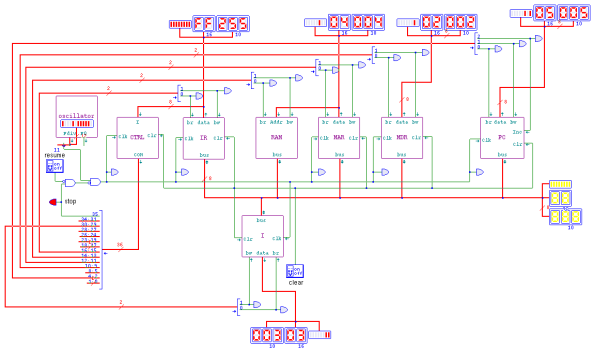


Figura u107.9. Situazione conclusiva del bus dati, dopo l'esecuzione delle istruzioni di caricamento. Video: *http://www.youtube.com/watch?v=AXUSrH49cF49w*



Istruzioni «store»

Viene proposto un altro esempio di macrocodice, nel quale si sperimentano le istruzioni **`store_imm`** e **`store_reg`**:

```
begin macrocode @ 0
start:
        load_imm #data_1
        store_imm #data_2
        move_mdr_i
        store_reg
stop:
        stop
data_1:
       .byte 15
data_2:
       .byte 0
end
```

In questo caso, si carica nel registro ***MDR*** il valore contenuto in memoria in corrispondenza dell'etichetta '**`data_1:`**'; quindi si memorizza, in corrispondenza della posizione di memoria corrispondente all'etichetta '**`data_2:`**', il valore contenuto in ***MDR*** (in pratica, in quella destinazione che prima conteneva il valore zero, viene copiato il valore 15, ovvero $0F_{16}$); quindi il contenuto del registro ***MDR*** viene copiato nel registro ***I*** e poi viene memorizzato il contenuto di ***MDR*** (che è rimasto sempre 15) nella posizione di memoria corrispondente al valore del registro ***I***. In pratica, alla fine si va a scrivere anche nella posizione 15 ($0F_{16}$) della memoria, e ci si mette il valore 15.

Figura u107.11. Contenuto della memoria RAM all'inizio dell'esecuzione.

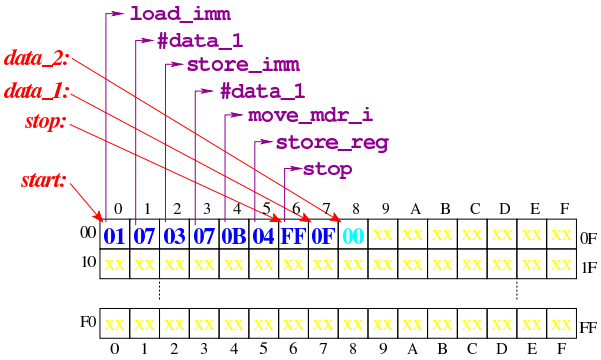


Figura u107.12. Contenuto della memoria RAM dopo l'esecuzione dell'istruzione **`store_imm`**.

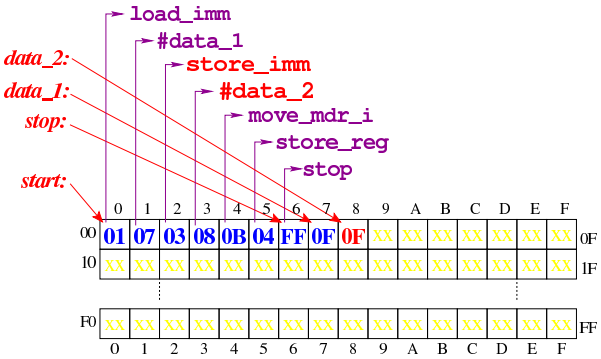


Figura u107.13. Contenuto della memoria RAM al termine dell'esecuzione.

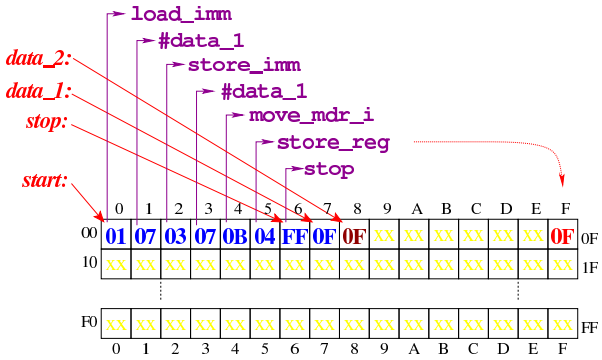
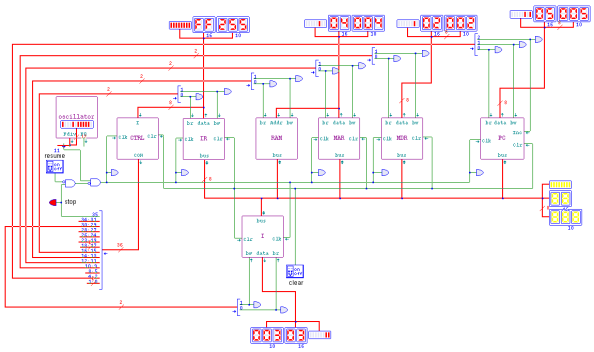
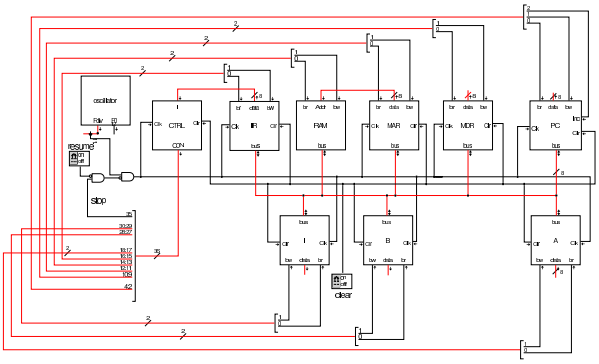


Figura u107.14. Situazione conclusiva del bus dati, dopo l'esecuzione delle istruzioni di memorizzazione. Video: *http://www.youtube.com/watch?v=lHxx3SR56hE56*



Nella terza versione della CPU dimostrativa, vengono aggiunti due registri che per il momento non hanno alcuno scopo particolare: ***A*** e ***B***. Tali registri sono realizzati nello stesso modo di ***I***, ***MDR***, ***MAR*** e ***IR***.

Figura u108.1. Il bus della CPU nella sua terza fase realizzativa.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti, i quali servono specificatamente a gestire i registri ***A*** e ***B***:

```
field a_br[17];          // A <-- bus
field a_bw[18];          // A --> bus
field b_br[27];          // B <-- bus
field b_bw[28];          // B --> bus
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op move_mdr_a {
  map move_mdr_a : 5;                  // move MDR to A
  +0[7:0]=5;
  operands op_0;
};
op move_a_mdr {
  map move_a_mdr : 6;                  // move A to MDR
  +0[7:0]=6;
  operands op_0;
};
op move_mdr_b {
  map move_mdr_b : 7;                  // move MDR to B
  +0[7:0]=7;
  operands op_0;
};
op move_b_mdr {
  map move_b_mdr : 8;                  // move B to MDR
  +0[7:0]=8;
  operands op_0;
};
```

```
begin microcode @ 0
…
move_mdr_a:
    a_br mdr_bw;                 // A <-- MDR
    ctrl_start ctrl_load;        // CNT <-- 0
//
move_a_mdr:
    mdr_br a_bw;                 // MDR <-- A
    ctrl_start ctrl_load;        // CNT <-- 0
//
move_mdr_b:
    b_br mdr_bw;                 // B <-- MDR
    ctrl_start ctrl_load;        // CNT <-- 0
//
move_b_mdr:
    mdr_br b_bw;                 // MDR <-- B
    ctrl_start ctrl_load;        // CNT <-- 0
…
end
```

Figura u108.5. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

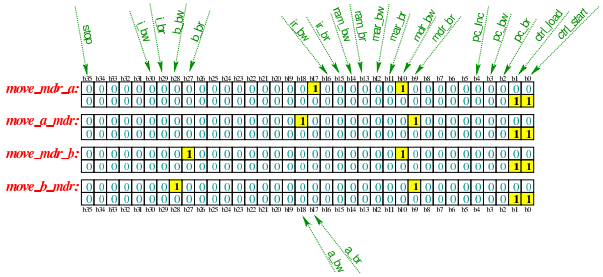


Tabella u108.6. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`move_mdr_a`	Copia il contenuto del registro ***MDR*** nel registro ***A***.
`move_a_mdr`	Copia il contenuto del registro ***A*** nel registro ***MDR***.
`move_mdr_b`	Copia il contenuto del registro ***MDR*** nel registro ***B***.
`move_b_mdr`	Copia il contenuto del registro ***B*** nel registro ***MDR***.

Come esempio viene proposto il macrocodice seguente:

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
stop:
        stop
data_1:
       .byte 17
data_2:
       .byte 11
end
```

In pratica, viene caricato nel registro ***MDR*** il valore corrispondente all'indirizzo in cui si trova l'etichetta '**`data_1:`**' (facendo i conti si tratta dell'indirizzo 7); successivamente, il valore di ***MDR*** viene copiato nel registro ***A***; quindi viene caricato nel registro ***MDR*** quanto contenuto nell'indirizzo di memoria corrispondente all'etichetta '**`data_2:`**' (indirizzo 8) e poi copiato nel registro ***B***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile all'indirizzo *allegati/circuiti-logici/scpu-sub-c.gm*

Figura u108.8. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

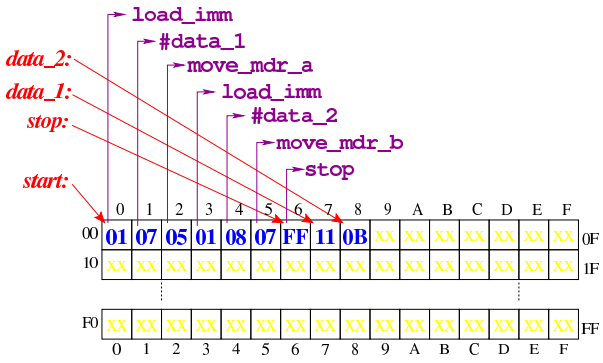
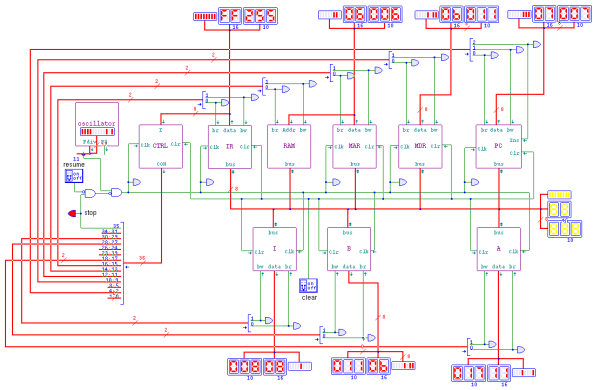


Figura u108.9. Situazione conclusiva del bus dati, dopo l'esecuzione delle istruzioni copia nei registri ***A*** e ***B***. Video: *http://www.youtube.com/watch?v=9qVsCKmxcdk*



Dalle istruzioni introdotte in questa versione della CPU dimostrativa, si può intendere che i dati contenuti nei registri possano essere copiati soltanto con la mediazione del registro ***MDR***; pertanto non esiste un'istruzione **`move_a_b`**. Questa è una semplificazione per evitare di dover dichiarare tante istruzioni nel macrocodice, ma in condizioni normali, tale scelta non sarebbe utile.

Istruzione «not» 846
Istruzione «and» 846
Istruzione «or» 847
Istruzione «xor» 848
Istruzioni «lshl» e «lshr» 849
Istruzioni «ashl» e «ashr» 850
Istruzioni «rotl» e «rotr» 851
Istruzione «add» 852
Istruzione «sub» 853

Nella quarta versione della CPU dimostrativa, viene aggiunta un'unità aritmetica, logica e di scorrimento (ALU), ma per il momento senza gestire gli indicatori (riporto, segno, zero e straripamento).

Figura u109.1. Il bus della CPU con l'aggiunta dell'unità ALU.

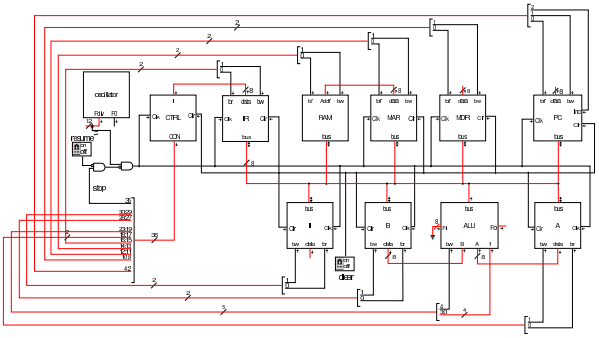


Figura u109.2. La struttura della ALU: si deve fare attenzione a non confondere le linee da un solo bit (di colore nero), rispetto a quelle che ne raccolgono in ranghi maggiori (di colore rosso).

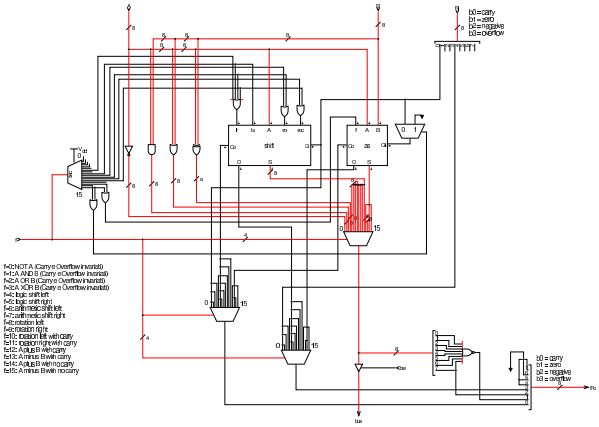


Figura u109.3. Modulo **shift** che si occupa di gestire gli scorrimenti e le rotazioni dei bit.

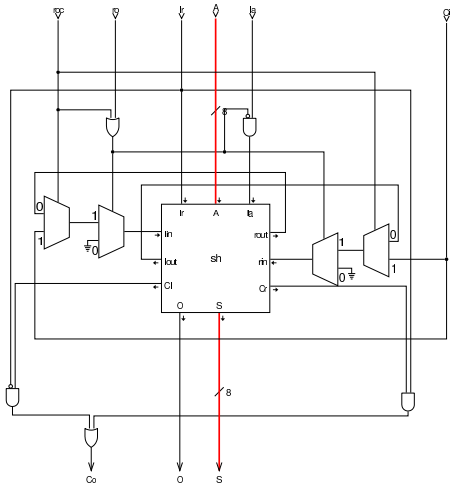


Figura u109.4. Modulo **sh**, contenuto nel modulo **shift**, per lo scorrimento dei bit.

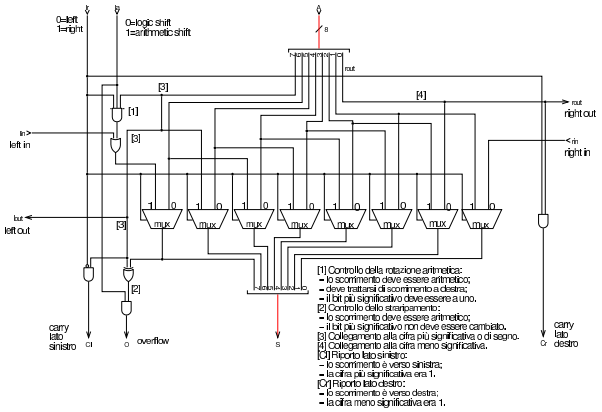
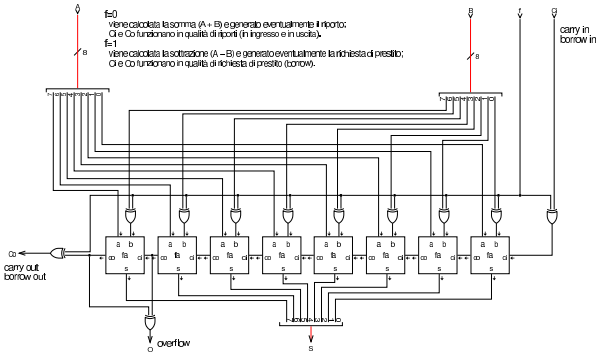


Figura u109.5. Modulo **as** della ALU che ha il compito di sommare o sottrarre gli ingressi.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti, i quali servono specificatamente a gestire la ALU. Si può osservare che la ALU ha il controllo di scrittura nel bus, ma non quello di lettura, dato che dal bus non riceve dati, e richiede il controllo della funzione che vi si vuole svolgere :

```
field alu_f[22:19]={
        not_a=0,
        a_and_b=1,
        a_or_b=2,
        a_xor_b=3,
        logic_shift_left=4,
        logic_shift_right=5,
        arith_shift_left=6,
        arith_shift_right=7,
        rotate_left=8,
        rotate_right=9,
        rotate_carry_left=10,
        rotate_carry_right=11,
        a_plus_b_carry=12,
        a_minus_b_borrow=13,
        a_plus_b=14,
        a_minus_b=15
};
field alu_bw[23];       // ALU --> bus
field fl_ar[24];        // FL <-- ALU
```


Tra i campi del bus di controllo si vede anche ***fl_ar*** che per ora può essere ignorato: viene chiarito il suo utilizzo quando nella prossima versione della CPU dimostrativa si aggiunge il registro ***FL***. Attualmente, nel microcodice vi si fa già riferimento, perché le microistruzioni prese ora in considerazione, in un secondo momento devono avere a che fare con tale registro.

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op not {
  map not : 32;                         // A = NOT A
  +0[7:0]=32;
  operands op_0;
};
op and {
  map and : 33;                         // A = A AND B
  +0[7:0]=33;
  operands op_0;
};
op or {
  map or : 34;                          // A = A OR B
  +0[7:0]=34;
  operands op_0;
};
op xor {
  map xor : 35;                         // A = A OR B
  +0[7:0]=35;
  operands op_0;
};
op lshl {
  map lshl : 36;                        // A = A << 1
  +0[7:0]=36;
  operands op_0;
};
op lshr {
  map lshr : 37;                        // A = A >> 1
  +0[7:0]=37;
  operands op_0;
};
op ashl {
  map ashl : 38;                        // A = A << 1
  +0[7:0]=38;
  operands op_0;
};
op ashr {
  map ashr : 39;                        // A = +/-A >> 1
  +0[7:0]=39;
  operands op_0;
};
op rotl {
  map rotl : 40;                        // A = A rotate left
  +0[7:0]=40;
  operands op_0;
};
op rotr {
  map rotr : 41;                        // A = A rotate right
  +0[7:0]=41;
  operands op_0;
};
op add {
  map add : 46;                         // A = A + B
  +0[7:0]=46;
  operands op_0;
};
op sub {
  map sub : 47;                         // A = A - B
  +0[7:0]=47;
```

```
  operands op_0;
};
```

```
begin microcode @ 0
…
not:
    a_br alu_f=not_a alu_bw fl_ar;        // A <-- NOT A
    ctrl_start ctrl_load;                 // CNT <-- 0
//
and:
    a_br alu_f=a_and_b alu_bw fl_ar;      // A <-- A AND B
    ctrl_start ctrl_load;                 // CNT <-- 0
//
or:
    a_br alu_f=a_or_b alu_bw fl_ar;       // A <-- A OR B
    ctrl_start ctrl_load;                 // CNT <-- 0
//
xor:
    a_br alu_f=a_xor_b alu_bw fl_ar;      // A <-- A XOR B
    ctrl_start ctrl_load;                 // CNT <-- 0
//
lshl:
    a_br alu_f=logic_shift_left alu_bw fl_ar; // A <-- A << 1
    ctrl_start ctrl_load;                    // CNT <-- 0
//
lshr:
    a_br alu_f=logic_shift_right alu_bw fl_ar; // A <-- A >> 1
    ctrl_start ctrl_load;                     // CNT <-- 0
//
ashl:
    a_br alu_f=arith_shift_left alu_bw fl_ar; // A <-- A*2
    ctrl_start ctrl_load;                    // CNT <-- 0
//
ashr:
    a_br alu_f=arith_shift_right alu_bw fl_ar; // A <-- A/2
    ctrl_start ctrl_load;                     // CNT <-- 0
//
rotl:
    a_br alu_f=rotate_left alu_bw fl_ar; // A <-- A rot. left
    ctrl_start ctrl_load;                // CNT <-- 0
//
rotr:
    a_br alu_f=rotate_right alu_bw fl_ar; // A <-- A rot. right
    ctrl_start ctrl_load;                 // CNT <-- 0
//
add:
    a_br alu_f=a_plus_b alu_bw fl_ar;     // A <-- A + B
    ctrl_start ctrl_load;                 // CNT <-- 0
//
sub:
    a_br alu_f=a_minus_b alu_bw fl_ar;    // A <-- A - B
    ctrl_start ctrl_load;                 // CNT <-- 0
…
end
```

Figura u109.9. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

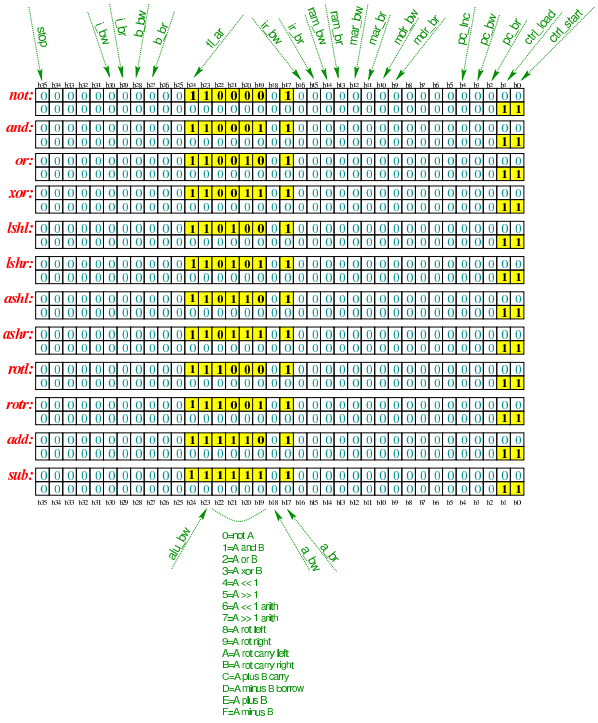


Tabella u109.10. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa. Nella descrizione sintetica delle operazioni si usa la notazione del linguaggio C.

Sintassi	Descrizione
`not`	***A*** = ~***A*** Complemento a uno del contenuto di ***A***.
`and`	***A*** = ***A*** & ***B*** Si assegna ad ***A*** il risultato di ***A*** AND ***B***, bit per bit.
`or`	***A*** = ***A*** | ***B*** Si assegna ad ***A*** il risultato di ***A*** OR ***B***, bit per bit.
`xor`	***A*** = ***A*** ^ ***B*** Si assegna ad ***A*** il risultato di ***A*** XOR ***B***, bit per bit.
`lshl`	***A*** = ***A*** << 1 Si assegna ad ***A*** il risultato dello scorrimento logico a sinistra dei bit di ***A***.
`lshr`	***A*** = ***A*** >> 1 Si assegna ad ***A*** il risultato dello scorrimento logico a destra dei bit di ***A***.
`ashl`	***A*** = ***A*** << 1 Si assegna ad ***A*** il risultato dello scorrimento aritmetico a sinistra dei bit di ***A*** (in pratica è identico a **`lshl`**).
`ashr`	***A*** = ***A*** >> 1 Si assegna ad ***A*** il risultato dello scorrimento aritmetico a destra dei bit di ***A***.
`rotl`	Si assegna ad ***A*** il risultato della rotazione a sinistra dei bit di ***A***.
`rotr`	Si assegna ad ***A*** il risultato della rotazione a destra dei bit di ***A***.
`add`	***A*** = ***A*** + ***B*** Si assegna ad ***A*** il risultato della somma di ***A*** e ***B***, senza tenere conto del riporto precedente.

Sintassi	Descrizione
`sub`	***A*** = ***A*** − ***B*** Si assegna ad ***A*** il risultato della sottrazione di ***A*** e ***B***, senza tenere conto della richiesta di prestito precedente.

Nelle sezioni successive, vengono proposti diversi esempi, nei quali si sperimentano tutte le istruzioni nuove introdotte.

Istruzione «not»

Listato u109.11. Macrocodice per sperimentare l'istruzione **`not`**: si carica un valore dalla memoria, lo si copia nel registro ***A***, si calcola il complemento a uno e il risultato va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-not.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        not
stop:
        stop
data_1:
        .byte 17
end
```

Figura u109.12. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

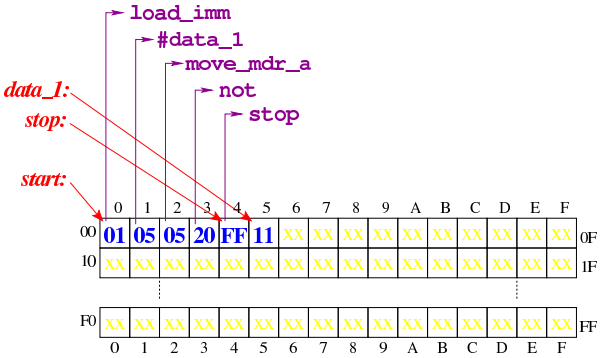
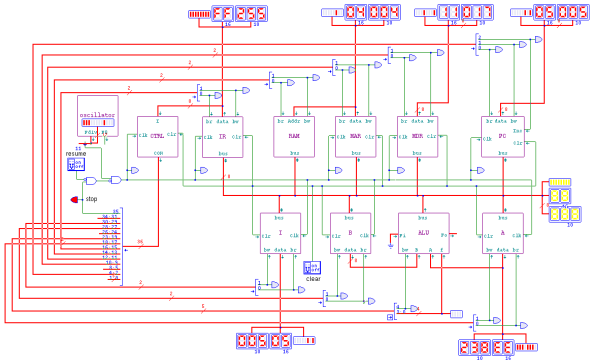


Figura u109.13. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **`not`**. Video: *http://www.youtube.com/watch?v=x5Vnhd72vh728*



Istruzione «and»

Listato u109.14. Macrocodice per sperimentare l'istruzione **and**: si caricano dalla memoria i valori da assegnare ai registri ***A*** e ***B***, quindi si esegue un AND binario che va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-and.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        and
stop:
        stop
data_1:
        .byte 17
data_2:
        .byte 11
end
```

Figura u109.15. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

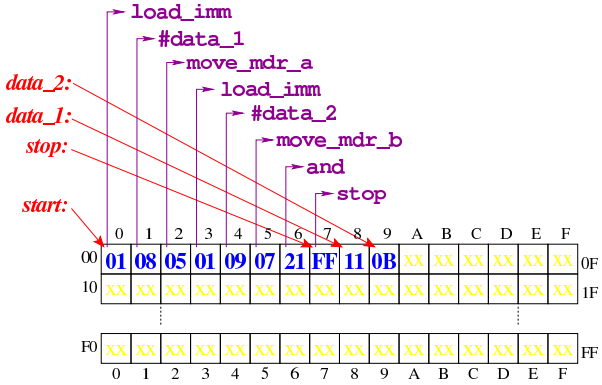
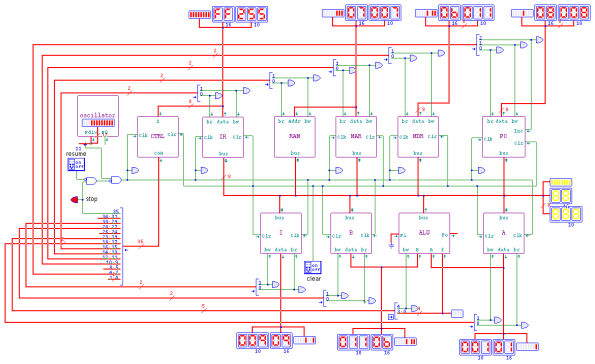


Figura u109.16. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **and**. Video: *http://www.youtube.com/watch?v=2ra7SHxBvYY*



Istruzione «or»

Listato u109.17. Macrocodice per sperimentare l'istruzione **`or`**: si caricano dalla memoria i valori da assegnare ai registri ***A*** e ***B***, quindi si esegue un OR binario che va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-or.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        or
stop:
        stop
data_1:
        .byte 17
data_2:
        .byte 11
end
```

Figura u109.18. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

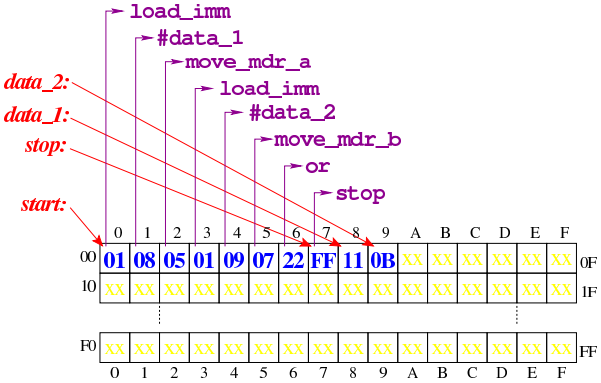
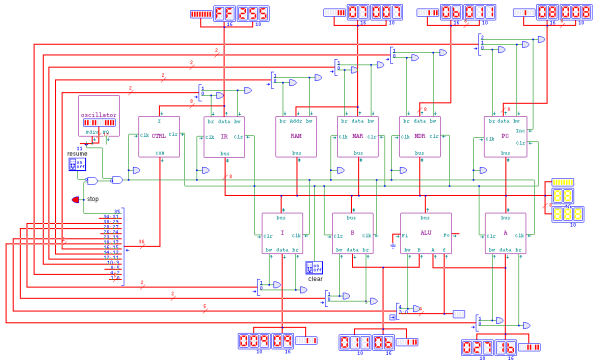


Figura u109.19. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **or**. Video: *http://www.youtube.com/watch?v=7E-2uA6fVoY*



Istruzione «xor»

Listato u109.20. Macrocodice per sperimentare l'istruzione **xor**: si caricano dalla memoria i valori da assegnare ai registri ***A*** e ***B***, quindi si esegue un XOR binario che va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-xor.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        xor
stop:
        stop
data_1:
        .byte 17
data_2:
        .byte 11
end
```


Figura u109.21. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

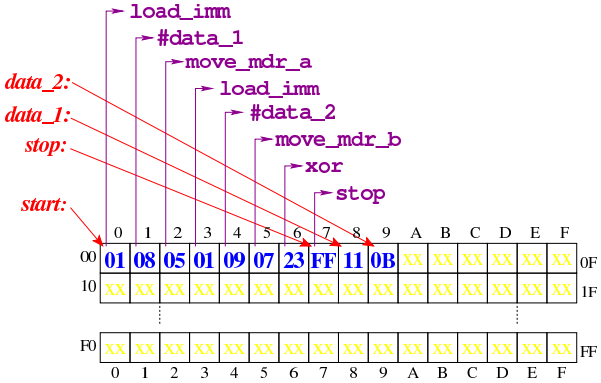
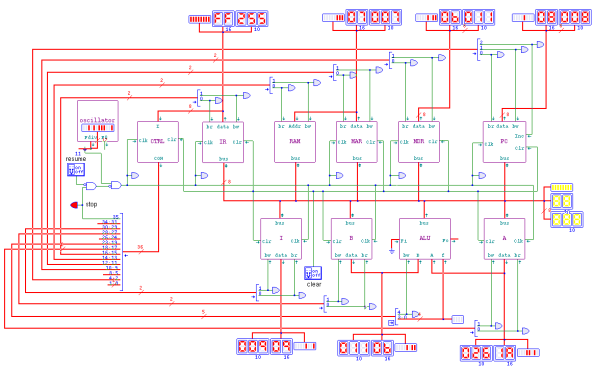


Figura u109.22. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **xor**. Video: *http://www.youtube.com/watch?v=TuzknbyeabQ*



Istruzioni «lshl» e «lshr»

Listato u109.23. Macrocodice per sperimentare le istruzioni di scorrimento logico: si carica un valore dalla memoria, lo si copia nel registro ***A***, si esegue lo scorrimento a sinistra e il risultato va ad aggiornare il registro ***A***; si copia il risultato nel registro ***B*** e si carica nuovamente il valore originale per eseguire lo scorrimento a destra (che va ad aggiornare sempre il registro ***A***). Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-lsh.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        lshl
        move_a_mdr
        move_mdr_b
        load_imm #data_1
        move_mdr_a
        lshr
stop:
        stop
data_1:
        .byte 17
end
```

Figura u109.24. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

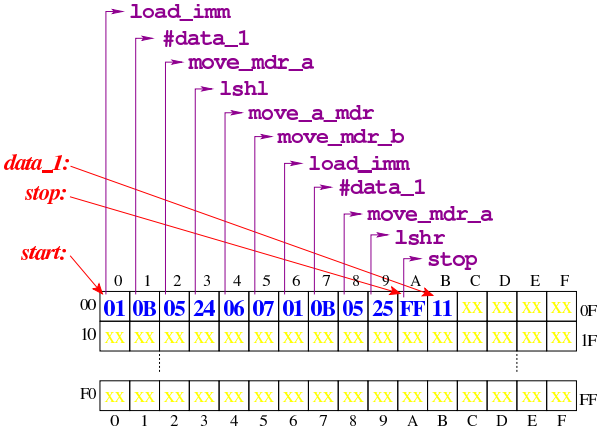
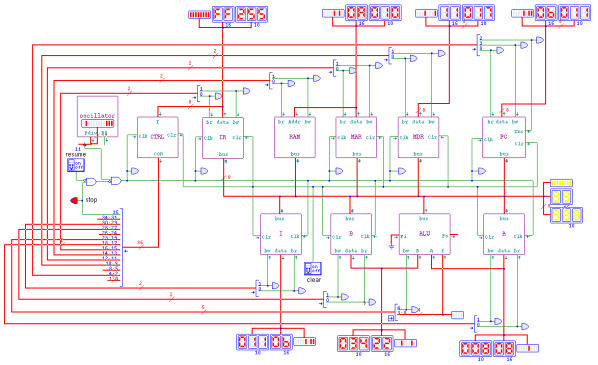


Figura u109.25. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=pkRfWYqGeB4*



Istruzioni «ashl» e «ashr»

Listato u109.26. Macrocodice per sperimentare le istruzioni di scorrimento aritmetico: si carica un valore dalla memoria, lo si copia nel registro ***A***, si esegue lo scorrimento a sinistra e il risultato va ad aggiornare il registro ***A***; si copia il risultato nel registro ***B*** e si carica nuovamente il valore originale per eseguire lo scorrimento a destra (che va ad aggiornare sempre il registro ***A***). Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-ash.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        ashl
        move_a_mdr
        move_mdr_b
        load_imm #data_1
        move_mdr_a
        ashr
stop:
        stop
data_1:
        .byte 143
end
```


Figura u109.27. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

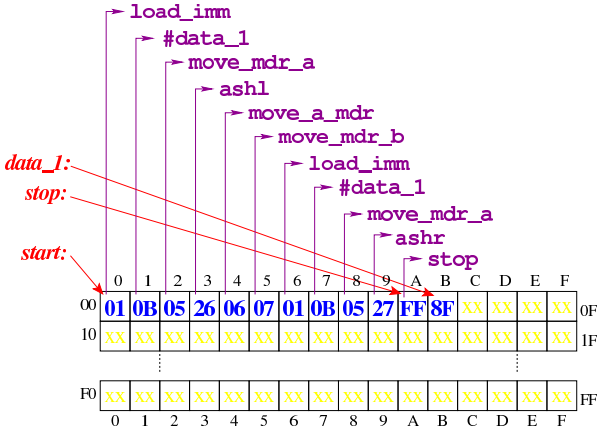
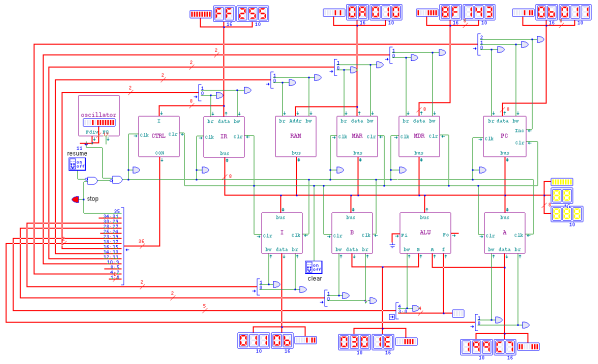


Figura u109.28. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=3rvR1WvWd1k*



Istruzioni «rotl» e «rotr»

Listato u109.29. Macrocodice per sperimentare le istruzioni di scorrimento logico: si carica un valore dalla memoria, lo si copia nel registro ***A***, si esegue la rotazione a sinistra e il risultato va ad aggiornare il registro ***A***; si copia il risultato nel registro ***B*** e si carica nuovamente il valore originale per eseguire la rotazione a destra (che va ad aggiornare sempre il registro ***A***). Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-rot.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        rotl
        move_a_mdr
        move_mdr_b
        load_imm #data_1
        move_mdr_a
        rotr
stop:
        stop
data_1:
        .byte 17
end
```

Figura u109.30. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

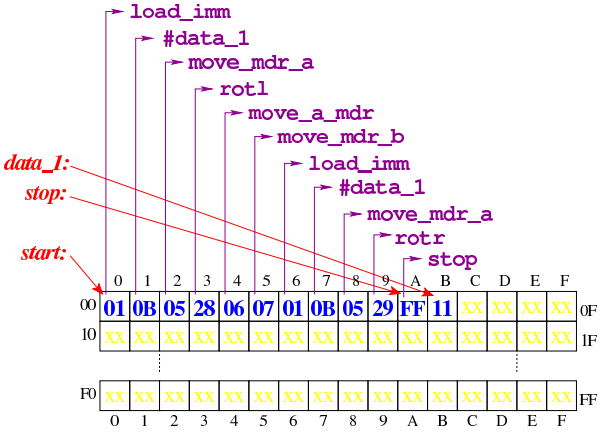
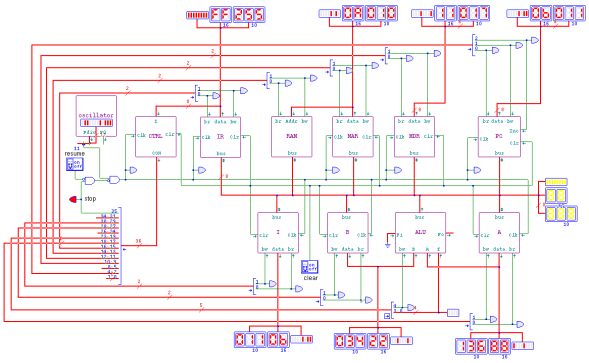


Figura u109.31. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=KCi8n6bnLQo*



Istruzione «add»

Listato u109.32. Macrocodice per sperimentare l'istruzione **add**: si caricano dalla memoria i valori da assegnare ai registri ***A*** e ***B***, quindi si esegue la somma che va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-add.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        add
stop:
        stop
data_1:
        .byte 17
data_2:
        .byte 11
end
```


Figura u109.33. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

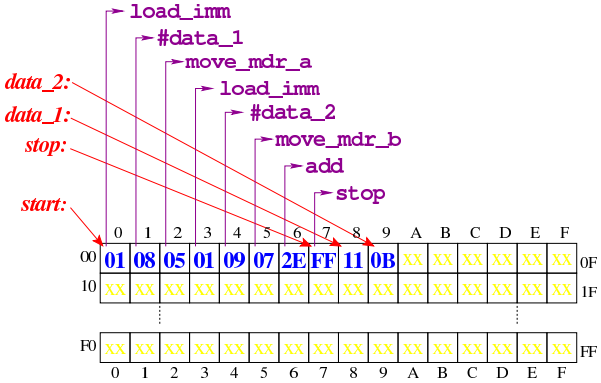
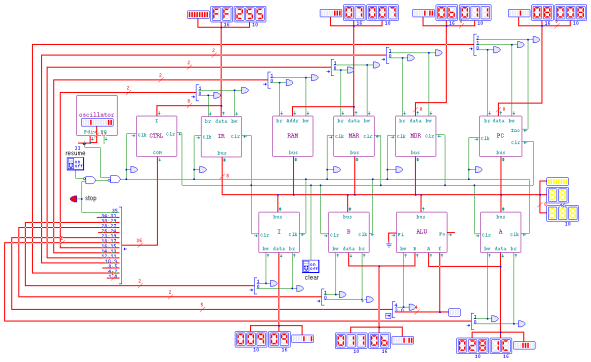


Figura u109.34. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **add**. Video: *http://www.youtube.com/watch?v=QQJwz2yVwA8*



Istruzione «sub»

Listato u109.35. Macrocodice per sperimentare l'istruzione **sub**: si caricano dalla memoria i valori da assegnare ai registri ***A*** e ***B***, quindi si esegue la sottrazione (***A*−*B***) che va ad aggiornare il registro ***A***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-sub.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        sub
stop:
        stop
data_1:
        .byte 17
data_2:
        .byte 11
end
```

Figura u109.36. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

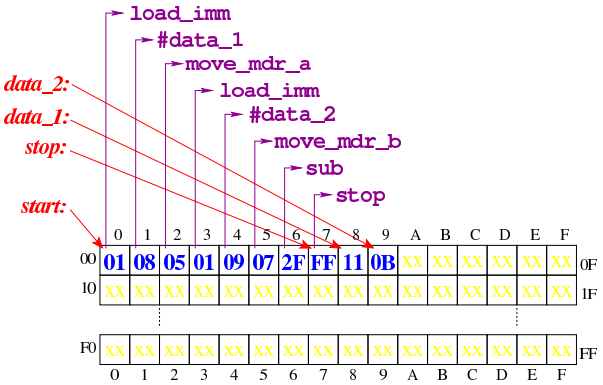
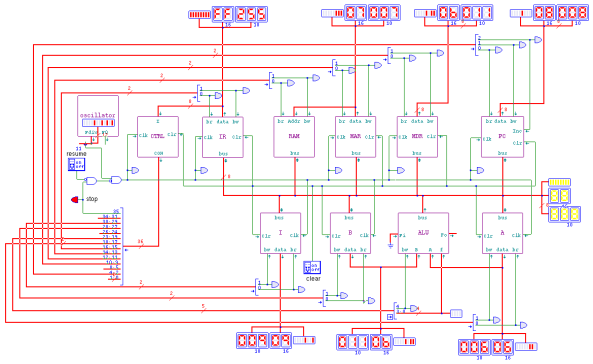


Figura u109.37. Situazione conclusiva del bus dati, dopo l'esecuzione dell'istruzione **sub**. Video: *http://www.youtube.com/watch?v=VRd8ilJbK_Y*

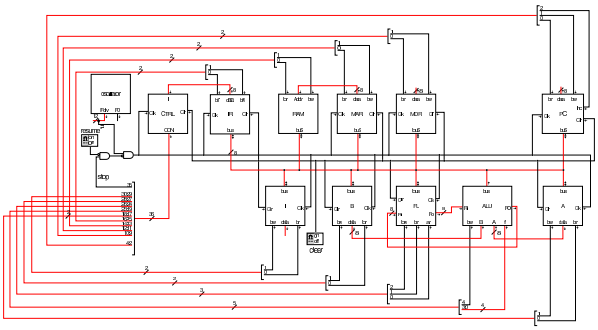


Versione E: indicatori

Istruzione «rotcl» e «rotcr» 857
Istruzione «add_carry» 858
Istruzione «sub_borrow» 860

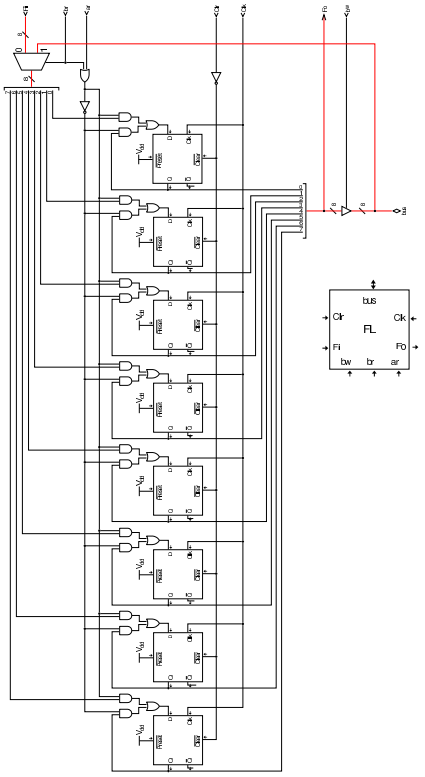
Nella quinta versione della CPU dimostrativa, viene aggiunto un registro per annotare lo stato degli indicatori, relativi all'esito di alcune operazioni svolte dalla ALU: riporto, segno, zero e straripamento.

Figura u110.1. Il bus della CPU con l'aggiunta del registro ***FL*** per la gestione degli indicatori.



Come si può comprendere dagli ingressi e dalle uscite che possiede, il registro ***FL*** può immettere dati nel bus e può essere modificato leggendo dati dal bus; inoltre, può leggere direttamente dalla ALU (ingresso ***Fi***), e per questo esiste un ingresso di abilitazione ulteriore, denominato ***ar*** (*ALU read*), mentre fornisce in ogni istante il proprio valore memorizzato alla ALU stessa (uscita ***Fo***).

Figura u110.2. La struttura interna del registro ***FL***: gli otto moduli che si vedono sono flip-flop D.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti (a parte ***fl_ar*** già apparso nella sezione precedente), i quali servono specificatamente a gestire il registro ***FL***:

```
field fl_ar[24];        // FL <-- ALU
field fl_br[25];        // FL <-- bus
field fl_bw[26];        // FL --> bus
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op move_mdr_fl {
  map move_mdr_fl : 9;          // move MDR to FL
  +0[7:0]=9;
  operands op_0;
};
op move_fl_mdr {
  map move_fl_mdr : 10;         // move FL to MDR
  +0[7:0]=10;
  operands op_0;
};
op rotcl {
  map rotcl : 42;               // A = A rotate carry left
  +0[7:0]=42;
  operands op_0;
};
op rotcr {
  map rotcr : 43;               // A = A rotate carry right
  +0[7:0]=43;
  operands op_0;
};
op add_carry {
  map add_carry : 44;           // A = A + B + carry
  +0[7:0]=44;
  operands op_0;
};
op sub_borrow {
  map sub_borrow : 45;          // A = A - B - borrow
  +0[7:0]=45;
  operands op_0;
};
```

```
begin microcode @ 0
…
//
move_mdr_fl:
    fl_br mdr_bw;               // FL <-- MDR
    ctrl_start ctrl_load;       // CNT <-- 0
//
move_fl_mdr:
    mdr_br fl_bw;               // MDR <-- FL
    ctrl_start ctrl_load;       // CNT <-- 0
//
rotcl:
    a_br alu_f=rotate_carry_left alu_bw fl_ar; // A <-- A rot. carry l
    ctrl_start ctrl_load;                     // CNT <-- 0
//
rotcr:
    a_br alu_f=rotate_carry_right alu_bw fl_ar; // A <-- A rot. carry r
    ctrl_start ctrl_load;                      // CNT <-- 0
//
add_carry:
    a_br alu_f=a_plus_b_carry alu_bw fl_ar; // A <-- A + B + carry
    ctrl_start ctrl_load;                  // CNT <-- 0
//
sub_borrow:
    a_br alu_f=a_minus_b_borrow alu_bw fl_ar; // A <-- A - B - borrow
    ctrl_start ctrl_load;                    // CNT <-- 0
…
end
```


Figura u110.6. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

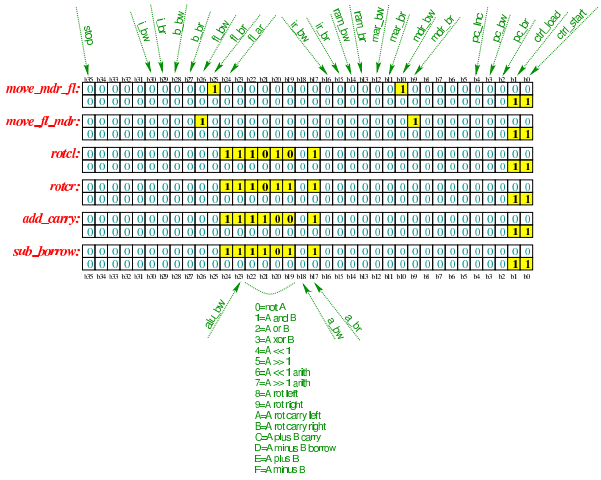


Tabella u110.7. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`move_mdr_fl`	Copia il contenuto del registro ***MDR*** nel registro ***FL***.
`move_fl_mdr`	Copia il contenuto del registro ***FL*** nel registro ***MDR***.
`rotcl`	Esegue la rotazione a sinistra del contenuto del registro ***A***, utilizzando anche l'indicatore di riporto.
`rotcr`	Esegue la rotazione a destra del contenuto del registro ***A***, utilizzando anche l'indicatore di riporto.
`add_carry`	Esegue la somma dei registri ***A*** e ***B***, tenendo conto del riporto precedente, aggiornando di conseguenza lo stesso registro ***A***.
`sub_carry`	Esegue la sottrazione ***A*−*B***, tenendo conto di un'eventuale richiesta di prestito precedente, aggiornando di conseguenza lo stesso registro ***A***.

Nelle sezioni successive, vengono proposti alcuni esempi, nei quali si sperimentano tutte le istruzioni nuove introdotte.

Istruzione «rotcl» e «rotcr»

Listato u110.8. Macrocodice per sperimentare le istruzioni **`rotcl`** e **`rotcr`**: si carica in memoria il valore da assegnare al registro ***A***, si eseguono cinque scorrimenti a sinistra, con l'uso del riporto e il risultato viene copiato nel registro ***B***; poi, con il valore presente in quel momento nel registro ***A***, si eseguono altri cinque rotazioni a destra, sempre con l'uso del riporto. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-rotc.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_1
        move_mdr_a
        rotcl
        rotcl
        rotcl
        rotcl
        rotcl
        move_a_mdr
        move_mdr_b
        rotcr
        rotcr
        rotcr
        rotcr
        rotcr
```

```
stop:
        stop
data_1:
        .byte 160
end
```

Figura u110.9. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

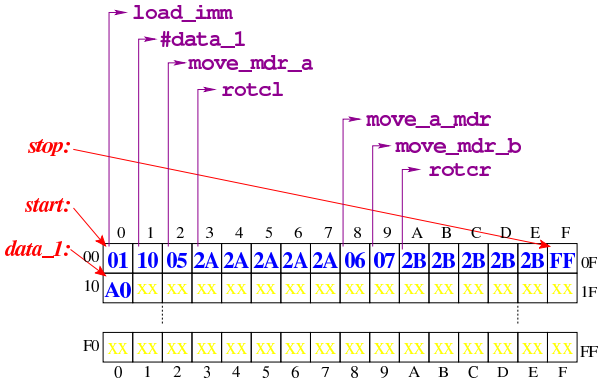
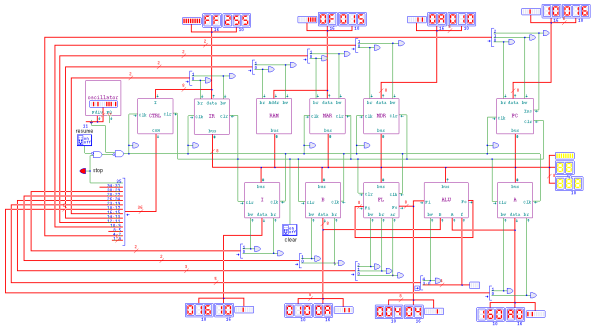


Figura u110.10. Situazione conclusiva del bus dati, dopo l'esecuzione delle istruzioni di rotazione con riporto. Video: *http://www.youtube.com/watch?v=Zl3d-Tg5C1Q*



Istruzione «add_carry»

«

Listato u110.11. Macrocodice per sperimentare l'istruzione **add_carry**: si vogliono sommare due numeri $12FF_{16}$ e $11EE_{16}$, necessariamente in due passaggi. Prima viene sommata la coppia FF_{16} e EE_{16}, con l'istruzione **add**, la quale produce il risultato ED_{16} con riporto, quindi viene sommata la coppia 12_{16} e 11_{16}, assieme al riporto precedente, ottenendo 24_{16}. In pratica, il risultato completo sarebbe $24ED_{16}$ che viene collocato in memoria dividendolo in due byte distinti. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-add_carry.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_0
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        add
        move_a_mdr
        store_imm #data_4
        load_imm #data_1
        move_mdr_a
        load_imm #data_3
        move_mdr_b
        add_carry
        move_a_mdr
        store_imm #data_5
stop:
        stop
// 0x12FF = 4863
data_0:
        .byte 0xFF
```



```
data_1:
        .byte 0x12
// 0x11EE = 4590
data_2:
        .byte 0xEE
data_3:
        .byte 0x11
data_4:
        .byte 0
data_5:
        .byte 0
end
```

Figura u110.12. Contenuto della memoria RAM prima dell'esecuzione. Le celle indicate con «xx» hanno un valore indifferente.

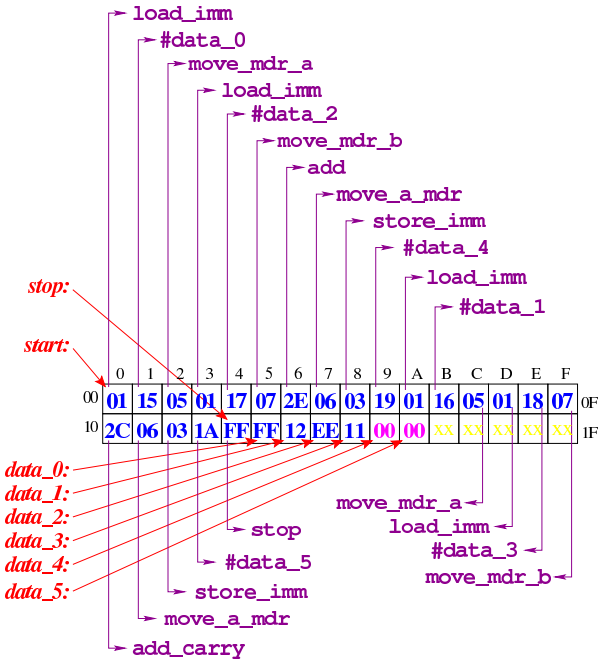


Figura u110.13. Al termine dell'esecuzione, le celle di memoria che devono contenere il risultato riportano il contenuto che si può vedere evidenziato qui.

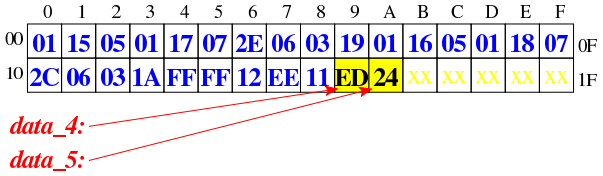
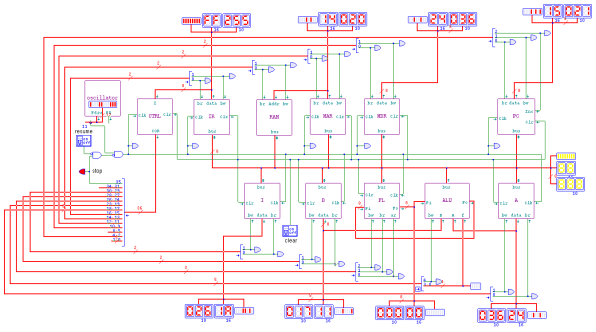


Figura u110.14. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=1Xu4MxWBwW4*



«

Listato u110.15. Macrocodice per sperimentare l'istruzione **sub_borrow**: si vuole eseguire la sottrazione $12EE_{16}-11FF_{16}$ e la si deve svolgere necessariamente in due passaggi. Prima viene sottratta la coppia EE_{16} e FF_{16}, con l'istruzione **sub**, la quale produce il risultato EF_{16} con richiesta di un prestito, quindi viene sottratta la coppia 12_{16} e 11_{16}, tenendo conto della richiesta del prestito dalle cifre precedenti, ottenendo 00_{16}. In pratica, il risultato completo sarebbe $00EF_{16}$ che viene collocato in memoria dividendolo in due byte distinti. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-sub_borrow.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_0
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        sub
        move_a_mdr
        store_imm #data_4
        load_imm #data_1
        move_mdr_a
        load_imm #data_3
        move_mdr_b
        sub_borrow
        move_a_mdr
        store_imm #data_5
stop:
        stop
// 0x12EE = 4846
data_0:
        .byte 0xEE
data_1:
        .byte 0x12
// 0x11FF = 4607
data_2:
        .byte 0xFF
data_3:
        .byte 0x11
data_4:
        .byte 0
data_5:
        .byte 0
end
```

Figura u110.16. Contenuto della memoria RAM prima dell'esecuzione. Le celle indicate con «xx» hanno un valore indifferente.

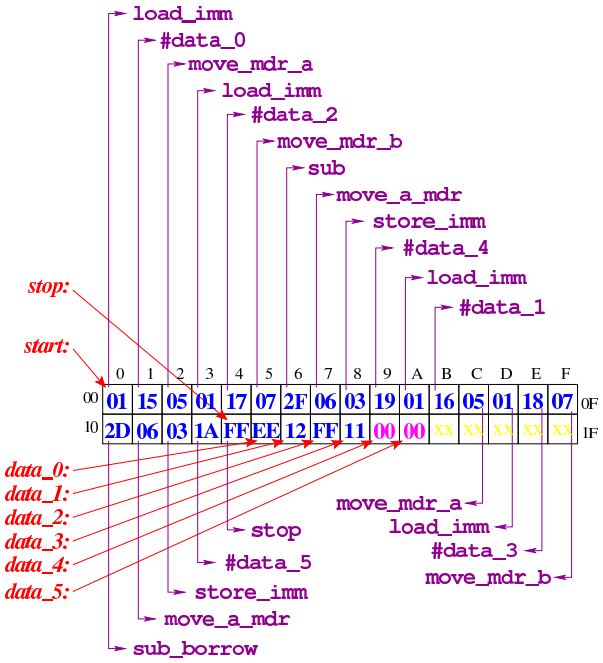


Figura u110.17. Al termine dell'esecuzione, le celle di memoria che devono contenere il risultato riportano il contenuto che si può vedere evidenziato qui.

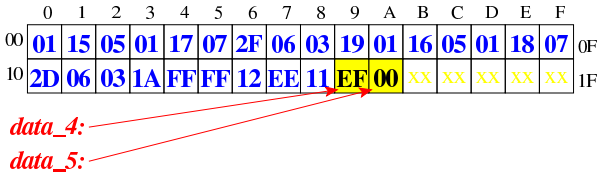
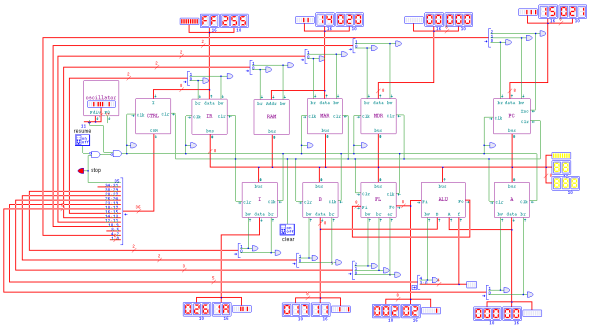
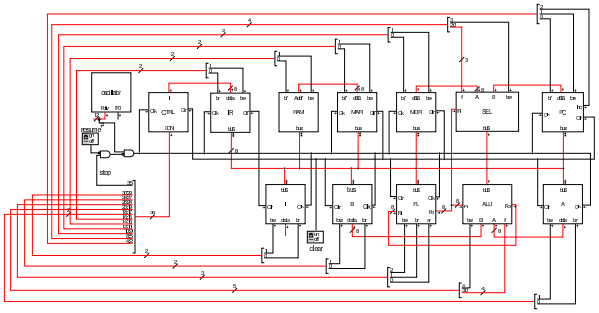


Figura u110.18. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=ofPUzdIids8*



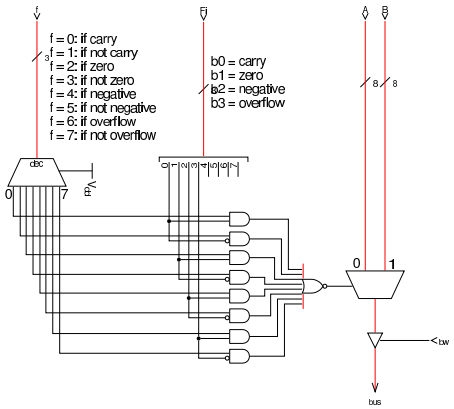
Nella sesta versione della CPU dimostrativa, viene aggiunto un modulo che consente di eseguire delle comparazioni, sulla base dello stato degli indicatori annotati nel registro ***FL***, scegliendo tra due valori che in questo progetto sono costituiti dal contenuto del registro ***PC*** o dal contenuto del registro ***MDR***.

Figura u111.1. Il bus della CPU con l'aggiunta del modulo ***SEL*** per la gestione condizioni.



Il modulo **SEL** riceve due valori dagli ingressi ***A*** e ***B***; dall'ingresso ***Fi*** riceve lo stato degli indicatori, così come emesso dal registro ***FL***. Sulla base della funzione che si seleziona attraverso l'ingresso ***f***, quando è attivo l'ingresso ***bw***, il modulo immette nel bus uno dei due valori disponibili negli ingressi ***A*** e ***B***. Quando la condizione rappresentata dalla funzione si avvera, viene scelto il valore dell'ingresso ***A***, altrimenti si prende ***B***.

Figura u111.2. La struttura interna del modulo **SEL**.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti, i quali servono specificatamente a gestire il modulo **SEL**:

```
field sel_f[7:5]={
        if_carry=0,
        if_not_carry=1,
        if_zero=2,
        if_not_zero=3,
        if_negative=4,
        if_not_negative=5,
        if_overflow=6,
        if_not_overflow=7
};
field sel_bw[8];        // SEL --> bus
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op jump_if_carry {
  map jump_if_carry : 16;        // jump to #nn if carry==1
  +0[7:0]=16;
  operands op_1;
```

```
};
op jump_if_not_carry {
  map jump_if_not_carry : 17;    // jump to #nn if carry==0
  +0[7:0]=17;
  operands op_1;
};
op jump_if_zero {
  map jump_if_zero : 18;         // jump to #nn if zero==1
  +0[7:0]=18;
  operands op_1;
};
op jump_if_not_zero {
  map jump_if_not_zero : 19;     // jump to #nn if zero==0
  +0[7:0]=19;
  operands op_1;
};
op jump_if_negative {
  map jump_if_negative : 20;     // jump to #nn if negative==1
  +0[7:0]=20;
  operands op_1;
};
op jump_if_not_negative {
  map jump_if_not_negative : 21; // jump to #nn if negative==0
  +0[7:0]=21;
  operands op_1;
};
op jump_if_overflow {
  map jump_if_overflow : 22;     // jump to #nn if overflow==1
  +0[7:0]=22;
  operands op_1;
};
op jump_if_not_overflow {
  map jump_if_not_overflow : 23; // jump to #nn if overflow==0
  +0[7:0]=23;
  operands op_1;
};
```

```
begin microcode @ 0
…
jump_if_carry:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_carry sel_bw         // PC = (carry ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_not_carry:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_not_carry sel_bw     // PC = (not_carry ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_zero:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_zero sel_bw          // PC = (zero ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_not_zero:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_not_carry sel_bw     // PC = (not_zero ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_negative:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_negative sel_bw      // PC = (negative ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_not_negative:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_not_negative sel_bw  // PC = (not_negative ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_overflow:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_overflow sel_bw      // PC = (overflow ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
//
jump_if_not_overflow:
    mar_br pc_bw;                       // MAR <-- PC
    pc_Inc;                             // PC++
```



```
    mdr_br ram_bw;                      // MDR <-- RAM[mar]
    pc_br sel_f=if_not_overflow sel_bw  // PC = (not_overflow ? MAR : PC)
    ctrl_start ctrl_load;               // CNT <-- 0
…
end
```

Figura u111.6. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

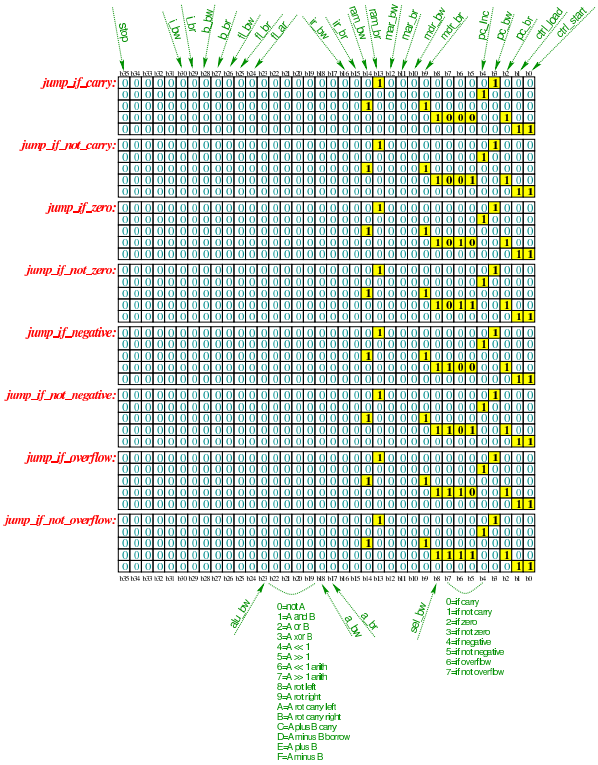


Tabella u111.7. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`jump_if_carry` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di riporto è attivo.
`jump_if_not_carry` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di riporto non è attivo.
`jump_if_zero` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore zero è attivo.
`jump_if_not_zero` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore zero non è attivo.
`jump_if_negative` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di valore negativo è attivo.
`jump_if_not_negative` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di valore negativo non è attivo.
`jump_if_overflow` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di straripamento è attivo.
`jump_if_not_overflow` ***indirizzo***	Salta all'indirizzo specificato se l'indicatore di straripamento non è attivo.

Listato u111.8. Macrocodice per sperimentare l'uso del modulo di selezione, nel quale si crea un ciclo che incrementa una variabile, di una unità alla volta, fino a quando questa variabile contiene il risultato della somma con un'altra. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-f-jump-if.gm* .

```
begin macrocode @ 0
start:
        load_imm #costante_zero
        move_mdr_b
```

```
        load_imm #variabile_x
        move_mdr_a
        add
ciclo:
        jump_if_zero #stop
        load_imm #variabile_y
        move_mdr_a
        load_imm #costante_uno
        move_mdr_b
        add
        move_a_mdr
        store_imm #variabile_y
        load_imm #variabile_x
        move_mdr_a
        load_imm #costante_uno
        move_mdr_b
        sub
        move_a_mdr
        store_imm #variabile_x
        jump #ciclo
stop:
        stop
costante_zero:
        .byte 0
costante_uno:
        .byte 1
variabile_x:
        .byte 3
variabile_y:
        .byte 7
end
```

Figura u111.9. Contenuto della memoria RAM prima dell'esecuzione. Le celle indicate con «xx» hanno un valore indifferente. Al termine dell'esecuzione, la cella di memoria all'indirizzo 23_{16}, corrispondente a ***variabile_y***, contiene il valore 10 ($0A_{16}$).

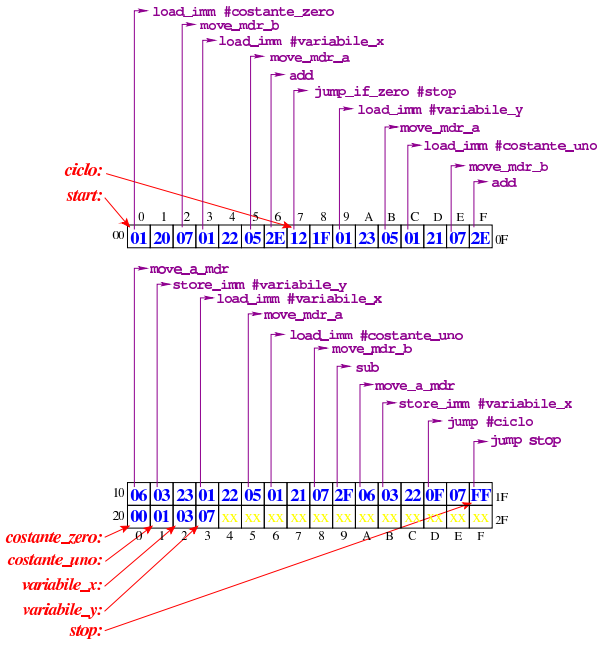
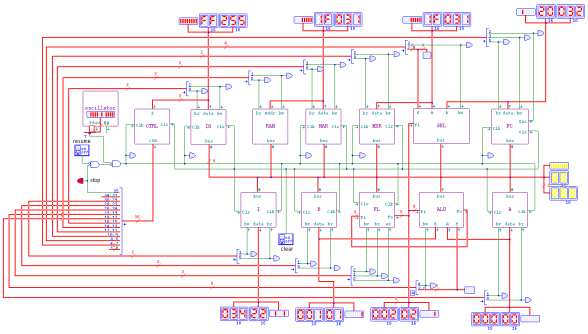


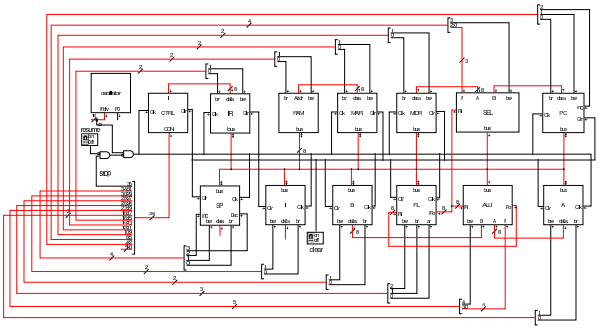
Figura u111.10. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=hFoOoGf86t0*



Istruzioni «push» e «pop» 873
Istruzioni «call» e «return» 873

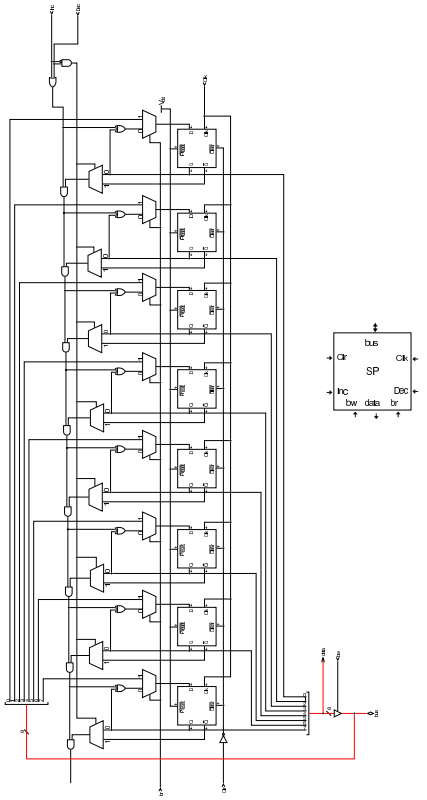
Nella settima versione della CPU dimostrativa, viene aggiunto il registro ***SP*** (*stack pointer*), utilizzato come indice per la pila dei dati. La pila serve principalmente a consentire le chiamate di procedure, tramite istruzioni **`call`** e **`return`**, oltre che a poter salvare e recuperare lo stato dei altri registri.

Figura u112.1. Il bus della CPU con l'aggiunta del registro ***SP*** per la gestione della pila.



Il registro ***SP*** ha due ingressi supplementari, ***Inc*** e ***Dec***, con lo scopo, rispettivamente, di incrementare o diminuire il valore memorizzato nel registro stesso, di una unità.

Figura u112.2. La struttura interna del registro **SP**.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti, i quali servono specificatamente a gestire il registro ***SP***:

```
field sp_br[31];        // SP <-- bus
field sp_bw[32];        // SP --> bus
field sp_Inc[33];       // SP++
field sp_Dec[34];       // SP--
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op call_imm {
  map call_imm : 24;                    // call #nn
  +0[7:0]=24;
  operands op_1;
};
op call_reg {
  map call_reg : 25;                    // call I
  +0[7:0]=25;
  operands op_0;
};
op return {
  map return : 26;                      // return
  +0[7:0]=26;
  operands op_0;
};
op push_mdr {
  map push_mdr : 27;                    // push MDR
  +0[7:0]=27;
  operands op_0;
};
op push_a {
  map push_a : 28;                      // push A
  +0[7:0]=28;
  operands op_0;
};
op push_b {
  map push_b : 29;                      // push B
  +0[7:0]=29;
  operands op_0;
};
op push_fl {
  map push_fl : 30;                     // push FL
  +0[7:0]=30;
  operands op_0;
};
op push_i {
  map push_i : 31;                      // push I
  +0[7:0]=31;
  operands op_0;
};
op pop_mdr {
  map pop_mdr : 48;                    // pop MDR
  +0[7:0]=48;
  operands op_0;
};
op pop_a {
  map pop_a : 49;                      // pop A
  +0[7:0]=49;
  operands op_0;
};
op pop_b {
  map pop_b : 50;                      // pop B
  +0[7:0]=50;
  operands op_0;
};
op pop_fl {
  map pop_fl : 51;                     // pop FL
  +0[7:0]=51;
  operands op_0;
};
op pop_i {
  map pop_i : 52;                      // pop I
  +0[7:0]=52;
  operands op_0;
};
```

```
begin microcode @ 0
…
call_imm:
    mar_br pc_bw;                   // MAR <-- PC
    pc_Inc;                         // PC++
    mdr_br ram_bw;                  // MDR <-- RAM[mar]
    sp_Dec;                         // SC--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br pc_bw;                   // RAM[mar] <-- PC
    pc_br mdr_bw;                   // PC <-- MDR
    ctrl_start ctrl_load;           // CNT <-- 0
//
call_reg:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br pc_bw;                   // RAM[mar] <-- PC
    pc_br i_bw;                     // PC <-- I
    ctrl_start ctrl_load;           // CNT <-- 0
```



```
//
return:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++;
    pc_br ram_bw;                   // PC <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
//
push_mdr:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br mdr_bw;                  // RAM[mar] <-- MDR
    ctrl_start ctrl_load;           // CNT <-- 0
//
push_a:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br a_bw;                    // RAM[mar] <-- A
    ctrl_start ctrl_load;           // CNT <-- 0
//
push_b:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br b_bw;                    // RAM[mar] <-- B
    ctrl_start ctrl_load;           // CNT <-- 0
//
push_fl:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br fl_bw;                   // RAM[mar] <-- FL
    ctrl_start ctrl_load;           // CNT <-- 0
//
push_i:
    sp_Dec;                         // SP--
    mar_br sp_bw;                   // MAR <-- SP
    ram_br i_bw;                    // RAM[mar] <-- I
    ctrl_start ctrl_load;           // CNT <-- 0
//
pop_mdr:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++
    mdr_br ram_bw;                  // MDR <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
//
pop_a:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++
    a_br ram_bw;                    // A <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
//
pop_b:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++
    b_br ram_bw;                    // B <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
//
pop_fl:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++
    fl_br ram_bw;                   // FL <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
//
pop_i:
    mar_br sp_bw;                   // MAR <-- SP
    sp_Inc;                         // SP++
    i_br ram_bw;                    // I <-- RAM[mar]
    ctrl_start ctrl_load;           // CNT <-- 0
…
end
```

Figura u112.6. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

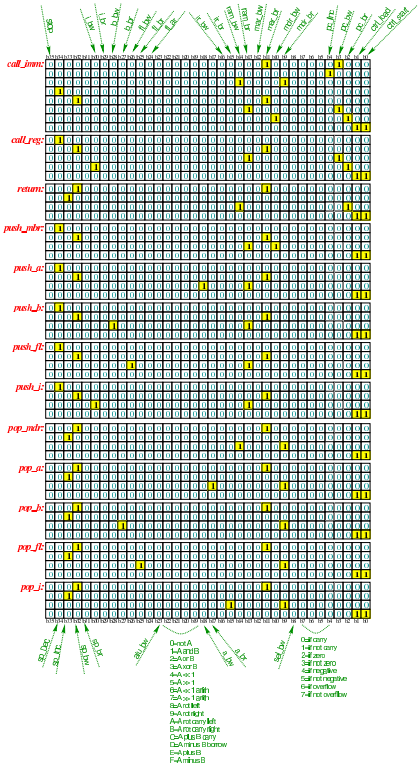


Tabella u112.7. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`call_imm` ***indirizzo***	Decrementa il registro ***SP*** e annota nella posizione di memoria a cui questo punta l'indirizzo dell'istruzione successiva, quindi aggiorna il registro ***PC*** con l'indirizzo indicato come argomento. In pratica esegue la chiamata di una procedura, annotando nella pila dei dati l'indirizzo di ritorno.
`call_reg`	Decrementa il registro ***SP*** e annota nella posizione di memoria a cui questo punta l'indirizzo dell'istruzione successiva, quindi aggiorna il registro ***PC*** con l'indirizzo contenuto nel registro ***I***. In pratica esegue la chiamata di una procedura, annotando nella pila dei dati l'indirizzo di ritorno.
`return`	Incrementa il registro ***SP*** e preleva l'indirizzo annotato in memoria, corrispondente all'indice della pila, quindi aggiorna il registro ***PC*** con tale valore. In pratica, consente di concludere una chiamata precedente.
`push_mdr`	Inserisce nella pila il valore del registro ***MDR***.
`push_a`	Inserisce nella pila il valore del registro ***A***.
`push_b`	Inserisce nella pila il valore del registro ***B***.
`push_fl`	Inserisce nella pila il valore del registro ***FL***.
`push_i`	Inserisce nella pila il valore del registro ***I***.
`pop_mdr`	Recupera dalla pila il valore del registro ***MDR***.
`pop_a`	Recupera dalla pila il valore del registro ***A***.

Sintassi	Descrizione
`pop_b`	Recupera dalla pila il valore del registro ***B***.
`pop_fl`	Recupera dalla pila il valore del registro ***FL***.
`pop_i`	Recupera dalla pila il valore del registro ***I***.

Istruzioni «push» e «pop»

Listato u112.8. Macrocodice per sperimentare l'uso delle istruzioni di inserimento ed estrazione di valori dalla pila dei dati: in questo caso, viene inserito nella pila il valore contenuto nel registro ***A*** e poi ripescato, ma nel registro ***B***. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-g-push-pop.gm* .

```
begin macrocode @ 0
start:
        load_imm #sp_bottom
        move_mdr_sp
        load_imm #data_0
        move_mdr_a
        push_a
        pop_b

stop:
        stop
sp_bottom:
        .byte 0x10
data_0:
        .byte 0xCC
end
```

Figura u112.9. Contenuto della memoria RAM: la cella nella posizione $0F_{16}$ viene scritta dall'istruzione **push_a**. Le celle indicate con «xx» hanno un valore indifferente.

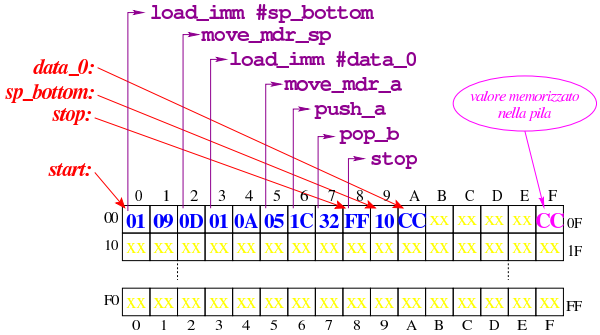
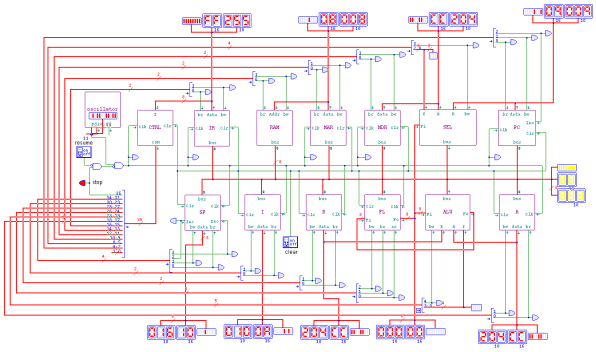


Figura u112.10. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=vAdVww3lD7I*



Istruzioni «call» e «return»

Listato u112.11. Macrocodice per sperimentare l'uso delle istruzioni di chiamata e ritorno dalle procedure. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-g-call-return.gm* .

```
begin macrocode @ 0
start:
        load_imm #sp_bottom
        move_mdr_sp
        call_imm #elabora
        move_a_mdr
        move_mdr_b
        jump #stop
elabora:
        load_imm #data_0
        move_mdr_a
        load_imm #data_1
        move_mdr_b
        add
        return
stop:
        stop
sp_bottom:
        .byte 0x20
data_0:
        .byte 0x0A
data_1:
        .byte 0x0B
end
```

Figura u112.12. Contenuto della memoria RAM: la cella nella posizione $1F_{16}$ viene scritta dall'istruzione **`call_imm`**. Le celle indicate con «xx» hanno un valore indifferente.

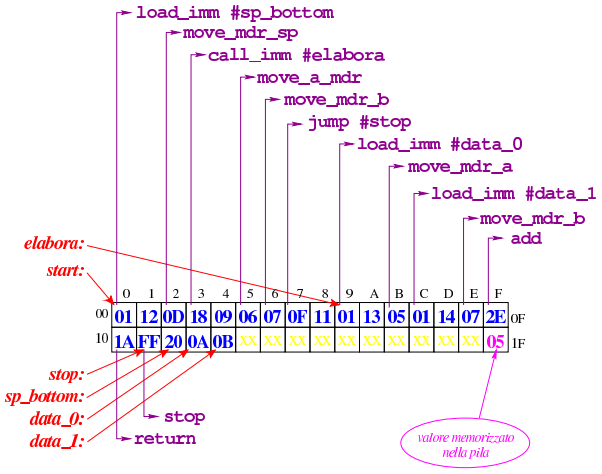
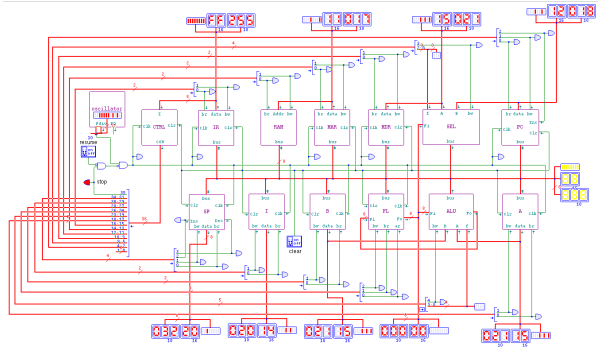


Figura u112.13. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=nWdXMvegkjc*



Versione H: I/O

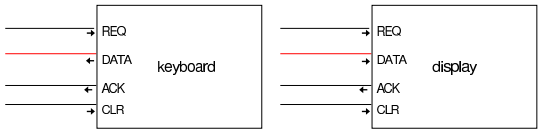
Generalizzazione della comunicazione con i dispositivi 875
Realizzazione dei dispositivi di I/O 876
Aspetto e funzionamento esteriore delle interfacce sincrone .. 878
Interfaccia sincrona della tastiera 879
Interfaccia sincrona dello schermo 880
Il bus della CPU con i dispositivi di I/O 881
Istruzione «out» 883
Istruzione «in» 884

La versione precedente è abbastanza completa per dimostrare le funzionalità principali di una CPU; in questa versione si passa a estendere il progetto iniziale, per consentire il collegamento con dispositivi di input-output. Per fare questo viene aggiunto un bus dati e un bus indirizzi per l'I/O (input-output), inoltre, si aggiungono alcune linee nel bus di controllo (***CON***), da utilizzare per i componenti che costituiscono l'interfaccia con i dispositivi di I/O. Il progetto della CPU dimostrativa si basa su Tkgate e nelle sezioni successive si mostra anche il codice utilizzato per realizzare i dispositivi in questione con questo simulatore.

Generalizzazione della comunicazione con i dispositivi

I dispositivi di I/O sono normalmente asincroni rispetto alla CPU (nel senso che non sono regolati dal clock che amministra la CPU), pertanto si rende necessario un protocollo per richiedere un'azione al dispositivo e per recepirne il risultato, tipico dei componenti asincroni. I dispositivi di I/O utilizzati in questo progetto dimostrativo hanno esteriormente le connessioni che si possono vedere nella figura successiva.

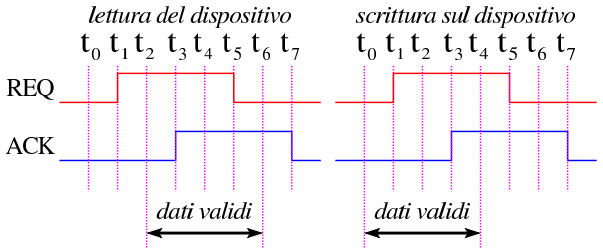
Figura u113.1. Un dispositivo di input e un dispositivo di output.



L'ingresso o l'uscita ***DATA*** consente di movimentare l'informazione che serve al dispositivo o che viene fornita dallo stesso. Gli ingressi ***REQ*** (*request*) e ***ACK*** (*acknowledge*) servono a negoziare il movimento dell'informazione (l'ingresso ***CLR*** serve ad azzerare il dispositivo). Vanno fatti due esempi, a seconda che si debba leggere un'informazione dal dispositivo, oppure che vi si voglia scrivere.

Per comunicare con un **dispositivo di input**, dal quale si deve leggere un'informazione, si comincia attivando l'ingresso ***REQ*** (t_1), con il quale si intende richiedere il dato. Il dispositivo riceve la richiesta e predispone l'uscita ***DATA*** con l'informazione (t_2); subito dopo, attiva l'uscita ***ACK*** (t_3). A questo punto, trovando attiva l'uscita ***ACK*** si può leggere il valore dall'uscita ***DATA*** (t_4) e al termine si può disattivare l'ingresso ***REQ*** (t_5). Il dispositivo, osservando la disattivazione dell'ingresso ***REQ*** sa che l'informazione è stata recepita, quindi smettere di fornire l'informazione richiesta (t_6) e disabilita a sua volta l'uscita ***ACK*** (t_7).

Figura u113.2. Fasi della lettura da un dispositivo di input e della scrittura su un dispositivo di output.



Per comunicare con un **dispositivo di output**, nel quale si deve scrivere un'informazione, si comincia fornendo l'informazione sull'ingresso ***DATA*** (t_0); subito dopo, si attiva l'ingresso ***REQ*** (t_1) per informare il dispositivo della disponibilità dell'informazione. Quindi il dispositivo recepisce l'informazione (t_2) e poi dà conferma attivando l'uscita ***ACK*** (t_3). Ricevendo la conferma, non è più necessario trattenere l'informazione disponibile nell'ingresso ***DATA*** (t_4), quindi viene disattivato l'ingresso ***REQ*** (t_5) e dopo di questo il dispositivo disattiva l'uscita ***ACK*** (t_7) concludendo l'operazione.

Realizzazione dei dispositivi di I/O

In questo progetto, per il momento, si realizzano soltanto due dispositivi di I/O: una tastiera e uno schermo. Dato che il progetto è sviluppato con Tkgate, i dispositivi vanno dichiarati attraverso codice Tcl/Tk. Il codice che viene mostrato qui è stato ottenuto modificando un dispositivo di esempio che fa parte della distribuzione di Tkgate.

Listato u113.3. File '`share/tkgate/vpd/kbd.tcl`' per simulare l'input di una tastiera. Il programma si limita a mostrare una piccola finestra vuota, selezionando la quale è possibile digitare da tastiera qualcosa che deve essere recepito dal dispositivo virtuale corrispondente.

```
VPD::register KBD
VPD::allow KBD::post

namespace eval KBD {
    # Dichiarazione delle variabili pubbliche. $kbd_w è un array
    # di cui si utilizza solo l'elemento $n, il quale identifica
    # univocamente l'istanza dell'interfaccia in funzione.
    variable kbd_w
    variable KD
    # Funzione richiesta da Tkgate per creare l'interfaccia.
    proc post {n} {
        variable kbd_w
        # Crea una finestra e salva l'oggetto in un elemento dell'array
        # $kbd_w.
        set kbd_w($n) [VPD::createWindow "KBD $n" -shutdowncommand "KBD::unpost $n"]
        # Collega la digitazione della tastiera, relativa all'oggetto
        # rappresentato da $kbd_w($n), alla funzione sendChar.
        bind $kbd_w($n) <KeyPress> "KBD::sendChar $n \"%A\""
        # Apre un canale di scrittura, denominato «KD».
        if {[info exists ::tkgate_isInitialized]} {
            VPD::outsignal $n.KD KBD::KD($n)
        }
    }
    # Funzione che recepisce la digitazione e la immette nel canale
    # denominato «KD», relativo all'istanza attuale dell'interfaccia.
    proc sendChar {n key} {
        variable KD
        if { [string length $key ] == 1 } {
            binary scan $key c c
            set KBD::KD($n) $c
        }
    }
    # Funzione richiesta da Tkgate per distruggere l'interfaccia.
    proc unpost {n} {
        variable kbd_w
        destroy $kbd_w($n)
        unset kbd_w($n)
    }
}
```

Listato u113.4. File '`share/tkgate/vpd/scr.tcl`' per simulare l'output su schermo a caratteri. Il programma mostra una finestra sulla quale possono poi apparire i caratteri trasmessi. Nel codice si fa riferimento al file '`textcurs.b`' che è già disponibile nella distribuzione di Tkgate.

```
image create bitmap txtcurs -file "$bd/txtcurs.b"

VPD::register SCR
VPD::allow SCR::post
VPD::allow SCR::data

namespace eval SCR {
    # Dichiarazione delle variabili pubbliche: si tratta di array
    # dei quali si utilizza solo l'elemento $n, il quale identifica
    # univocamente l'istanza dell'interfaccia in funzione.
    variable scr_w
    variable scr_pos
    # Funzione richiesta da Tkgate per creare l'interfaccia.
    proc post {n} {
        variable scr_w
        variable scr_pos
        # Crea una finestra e salva l'oggetto in un elemento dell'array
        # $scr_w.
        set scr_w($n) [VPD::createWindow "SCR $n" -shutdowncommand "SCR::unpost $n"]
        # Per maggiore comodità, copia il riferimento all'oggetto nella
        # variabile locale $w e in seguito fa riferimento all'oggetto
        # attraverso questa seconda variabile.
        set w $scr_w($n)
        text $w.txt -state disabled
        pack $w.txt
        # Mette il cursore alla fine del testo visualizzato.
        $w.txt image create end -image txtcurs
        # Apre un canale di lettura, denominato «RD», associandolo
        # alla funzione «data».
        if {[info exists ::tkgate_isInitialized]} {
            VPD::insignal $n.RD -command "SCR::data $n" -format %d
        }
        # Azzera il contatore che tiene conto dei caratteri visualizzati
        # sullo schermo.
        set scr_pos($n) 0
    }
    # Funzione richiesta da Tkgate per distruggere l'interfaccia.
    proc unpost {n} {
        variable scr_w
        destroy $scr_w($n)
        unset scr_w($n)
    }
    # Funzione usata per recepire i dati da visualizzare.
    proc data {n c} {
        variable scr_w
        variable scr_pos
        # Per maggiore comodità, copia il riferimento all'oggetto che
        # rappresenta l'interfaccia nella variabile $w.
        set w $scr_w($n)
        catch {
            # La variabile $c contiene il carattere da visualizzare.
            if { $c == 7 } {
                # BEL
                bell
                return
            } elseif { $c == 127 || $c == 8 } {
                # DEL | BS
                if { $scr_pos($n) > 0 } {
                    # Cancella l'ultimo carattere visualizzato, ma solo
                    # se il contatore dei caratteri è maggiore di zero,
                    # altrimenti sparirebbe il cursore e la
                    # visualizzazione verrebbe collocata in un'area
                    # non visibile dello schermo.
                    $w.txt configure -state normal
                    $w.txt delete "end - 3 chars"
                    $w.txt see end
                    $w.txt configure -state disabled
                    set scr_pos($n) [expr {$scr_pos($n) - 1}]
                }
                return
            } elseif { $c == 13 } {
                # CR viene trasformato in LF.
                set c 10
            }
            # Converte il numero del carattere in un simbolo
            # visualizzabile.
            set x [format %c $c]
            # Visualizza il simbolo.
            $w.txt configure -state normal
            $w.txt insert "end - 2 chars" $x
            $w.txt see end
            $w.txt configure -state disabled
            # Aggiorna il contatore dei caratteri visualizzati.
            set scr_pos($n) [expr {$scr_pos($n) + 1}]
        }
    }
}
```

I due programmi Tcl/Tk servono a fornire due moduli software, a cui poi si fa riferimento attraverso del codice Verilog. Nei listati successivi si vedono i moduli **`keyboard`** e **`display`** che graficamente si mostrano esattamente come nella figura u113.1.

Listato u113.5. Codice Verilog per il modulo **`keyboard`**.

```
module keyboard(DATA, REQ, ACK, CLR);
output ACK;
output [7:0] DATA;
input REQ;
input CLR;
reg ready;
reg [7:0] key;

  initial
    begin
```

```
      ready = 0;
      key = 0;
    end

  always
    begin
      @(posedge CLR)
      ready = 0;
      key = 0;
    end

  initial $tkg$post("KBD","%m");

  always
    begin
      @ (posedge REQ);
      # 5;
      key = $tkg$recv("%m.KD");
      # 5;
      ready = 1'b1;
      # 5;
      @ (negedge REQ);
      # 5;
      ready = 1'b0;
    end

  assign DATA = key;
  assign ACK = ready;

endmodule
```

Listato u113.6. Codice Verilog per il modulo **`display`**.

```
module display(DATA, REQ, ACK, CLR);
output ACK;
input [7:0] DATA;
input REQ;
input CLR;
reg ready;

  initial
    begin
      ready = 0;
    end

  initial $tkg$post("SCR","%m");

  always
    begin
      @(posedge CLR)
      ready = 0;
    end

  always
    begin
      @(posedge REQ);
      # 5;
      $tkg$send("%m.RD",DATA);
      # 5;
      ready = 1'b1;
      # 5;
      @(negedge REQ);
      # 5;
      ready = 1'b0;
    end

  assign ACK = ready;

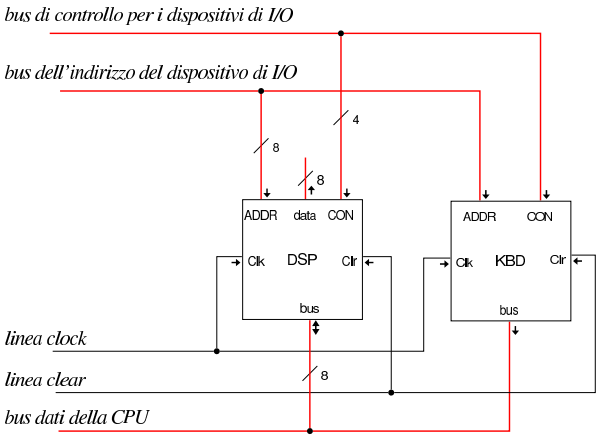
endmodule
```

Fino a qui, i dispositivi descritti funzionano in modo asincrono, ma per essere utilizzati devono essere adattati per poter funzionare in modo sincrono.

Aspetto e funzionamento esteriore delle interfacce sincrone

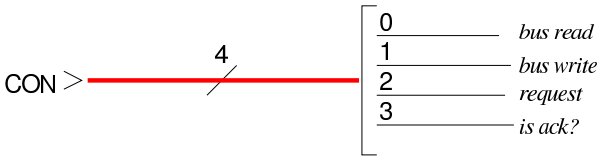
Le interfacce sincrone dei dispositivi di I/O devono potersi collegare al bus della CPU come gli altri moduli già presenti; tuttavia, per semplificare il cablaggio, invece di disporre di linee di controllo personali, viene creato un bus aggiuntivo, in cui indicare l'indirizzo del modulo a cui si vuole fare riferimento.

Figura u113.7. Connessione dei moduli di I/O.



Come si vede dal disegno, il bus connesso agli ingressi ***ADDR*** serve a selezionare il dispositivo, mentre il bus connesso agli ingressi ***CON*** serve a uniformare le linee di controllo per tutti i moduli di I/O. In pratica, viene mostrato in seguito che questi due bus aggiuntivi provengono dallo stesso bus di controllo complessivo.

Gli ingressi ***CON*** sono uniformi, ma ogni modulo utilizza solo ciò che gli serve, ignorando il resto:

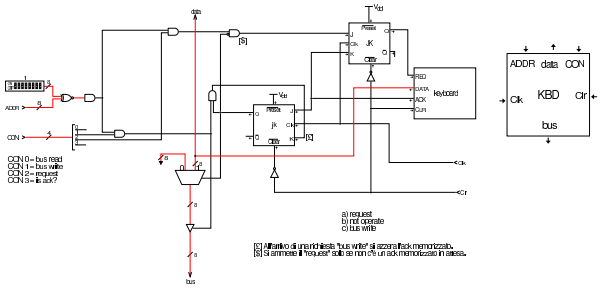


Le linee ***bus read*** e ***bus write*** sono le stesse degli altri moduli già descritti, riferendosi all'ordine di leggere o di scrivere sul bus della CPU. La linea ***request*** serve a richiedere l'operazione di lettura o scrittura del dispositivo, ma senza la necessità di mantenerla attiva come nel protocollo di comunicazione asincrona. La linea ***is ack?*** serve a ottenere, in qualche modo, l'informazione sul fatto che sia stata ricevuta la conferma da parte del dispositivo.

Interfaccia sincrona della tastiera

Il modulo di interfaccia della tastiera utilizza solo due delle quattro linee di controllo: ***bus write*** e ***request***.

Figura u113.9. Modulo **`KBD`** che collega la tastiera al bus della CPU.



Per comunicare con il modulo della tastiera, è necessario inizialmente un segnale ***request*** che all'arrivo dell'impulso di clock viene memorizzato nel flip-flop JK superiore, attivandolo; tale flip-flop ha il compito di trasferire e fissare tale valore sull'ingresso ***REQ*** del dispositivo. Il secondo flip-flop JK, in posizione centrale, ha invece il compito di memorizzare l'esito emesso dall'uscita ***ACK*** e, se tale valore risulta memorizzato, non permette la ricezione di una nuova richiesta. Dopo la ricezione di un segnale ***request*** acquisito correttamente, nella migliore delle ipotesi, il dispositivo ha già accumulato

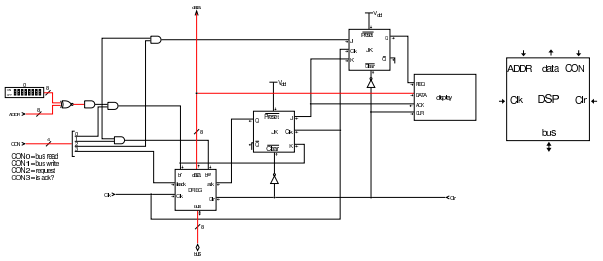
un carattere digitato da tastiera e risponde quasi subito mettendo tale valore nella sua uscita ***DATA*** e poi attivando la sua uscita ***ACK***. Al secondo impulso di clock, il flip-flop che manteneva attivo l'ingresso ***REQ*** si azzera e invece si attiva il secondo flip-flop al centro del disegno; tuttavia, il dispositivo mantiene il valore dell'uscita ***DATA***. A questo punto si riceve il segnale ***bus write*** che consente di immettere il valore ricevuto dalla tastiera nel bus della CPU, azzerando contestualmente il flip-flop centrale. Se invece non si riesce a ottenere un carattere dalla tastiera, nel tempo stabilito, si ottiene il valore nullo (00_{16}), dato che manca l'attivazione del flip-flop che conserva lo stato di ***ACK***.

A livello di macrocodice, se la lettura della tastiera produce un carattere nullo, significa che non c'è alcun carattere pronto e occorre ripetere la lettura.

Interfaccia sincrona dello schermo

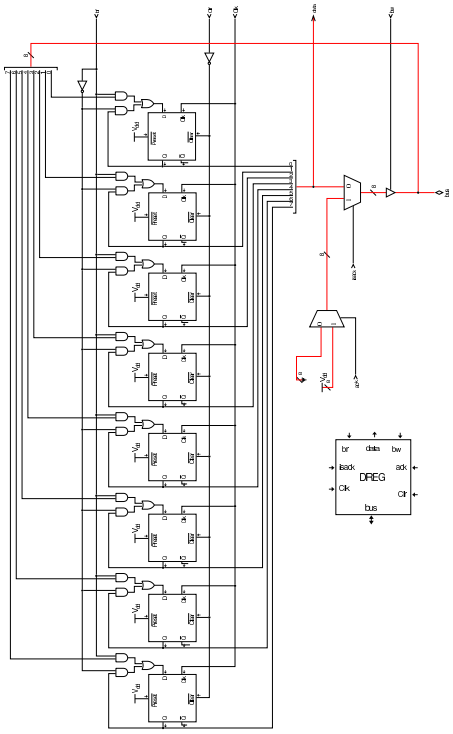
Il modulo **DSP** utilizza tutte le linee dell'ingresso di controllo, in quanto deve poter leggere dal bus della CPU, quando si vuole visualizzare un carattere sullo schermo, ma deve anche poter scrivere sul bus, per fornire un codice di conferma della riuscita della visualizzazione.

Figura u113.10. Modulo **DSP** che collega lo schermo al bus della CPU.



Il modulo **DSP** utilizza un registro modificato che serve principalmente per memorizzare il carattere da rappresentare sullo schermo; tuttavia, quando si attiva il suo ingresso ***isack***, può immettere nel bus della CPU l'esito della visualizzazione: 00_{16} vuol dire che questa non è ancora stata confermata, mentre FF_{16} indica una rappresentazione avvenuta correttamente.

Figura u113.11. Registro **DREG** che accumula il carattere da visualizzare.



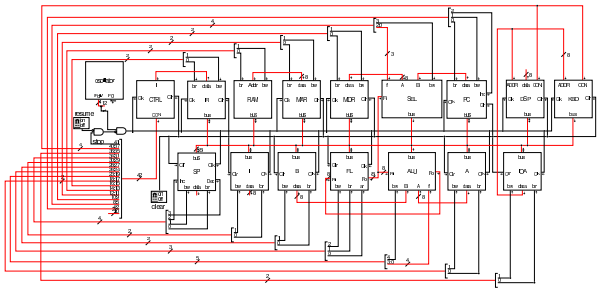
Per visualizzare un carattere sullo schermo, si comincia attivando la linea ***bus read***: all'arrivo dell'impulso di clock il registro accumula il carattere leggendolo dal bus della CPU, mentre il flip-flop JK centrale si azzera, azzerando così l'ingresso ***ack*** del registro ***DREG***. Quindi deve essere attivata la linea ***request*** e all'arrivo dell'impulso di clock questo valore viene memorizzato nel flip-flop JK superiore, il quale attiva così l'ingresso ***REQ*** del dispositivo. A quel punto, ritardo di propagazione permettendo, il dispositivo mostra il carattere già presente nel suo ingresso ***DATA*** (proveniente dal registro ***DREG*** e a un certo punto risponde attivando la sua uscita ***ACK***. Quando l'uscita ***ACK*** del dispositivo si attiva e sopraggiunge un impulso di clock, il registro che manteneva il segnale ***REQ*** si azzera, mentre si attiva il flip-flop JK centrale, attivando di conseguenza l'ingresso ***ack*** del registro ***DREG***.

Per la visualizzazione di un carattere, sono sufficienti i due cicli di clock iniziali, ma per verificare che la visualizzazione sia avvenuta effettivamente, occorre intervenire nuovamente con un'interrogazione. In tal caso si attiva la linea ***is ack?*** e ***bus write***, in modo da immettere nel bus della CPU il valore che può essere 00_{16} o FF_{16}, a seconda del fatto che non sia ancora stata ottenuta la conferma oppure che invece questa ci sia stata.

Il bus della CPU con i dispositivi di I/O

Nel bus della CPU, oltre ai moduli dei dispositivi di I/O, si aggiunge un registro, denominato ***IOA***, con lo scopo di conservare l'indirizzo del dispositivo di I/O con il quale si vuole comunicare.

Figura u113.12. Il bus della CPU con l'aggiunta del registro ***IOA*** e dei moduli di I/O.



Dal disegno si può vedere che il bus di controllo complessivo è stato modificato, inserendo delle linee per il controllo del registro ***IOA*** e le quattro linee necessarie a controllare i dispositivi di I/O, spostando di conseguenza la linea usata per fermare il segnale di clock. Pertanto, nel codice della dichiarazione delle memorie e in quello che descrive i campi del bus di controllo, si apportano le modifiche seguenti:

```
map       bank[7:0]   ctrl.m0;
microcode bank[31:0]  ctrl.m1;
microcode bank[41:32] ctrl.m2;
macrocode bank[7:0]   ram.m3;
…
field ioa_br[35];        // IOA <-- bus
field ioa_bw[36];        // IOA --> bus
field io_br[37];         // I/O <-- bus
field io_bw[38];         // I/O --> bus
field io_req[39];        // I/O request
field io_isack[40];      // I/O is ack?
field stop[41];          // stop clock
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice; in particolare, si ammettono macroistruzioni che richiedono due argomenti:

```
operands op_2 {
    //
    // [........][mmmmmmmm][nnnnnnnn]
    //
    #1,#2 = { +1=#1[7:0]; +2=#2[15:8]; };
};
…
op in {
  map in : 48;                    // read input from I/O bus
  +0[7:0]=48;
  operands op_1;
};
op out {
  map out : 49;                   // write output to I/O bus
  +0[7:0]=49;
  operands op_1;
};
op io_is_ack {
  map io_is_ack : 50;
  +0[7:0]=50;
  operands op_2;
};
```

```
begin microcode @ 0
…
in:
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    mdr_br ram_bw;              // MDR <-- RAM[mar]
    ioa_br mdr_bw;              // IOA <-- MDR
    io_req;                     //
    io_br;                      // non fa alcunché
    a_br io_bw;                 // A <-- I/O
    ctrl_start ctrl_load;       // CNT <-- 0

out:
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    mdr_br ram_bw;              // MDR <-- RAM[mar]
    ioa_br mdr_bw;              // IOA <-- MDR
    io_br a_bw;                 // IO <-- A
    io_req;                     //
    ctrl_start ctrl_load;       // CNT <-- 0

io_is_ack:
    mar_br pc_bw;               // MAR <-- PC
```



```
    pc_Inc;                     // PC++
    mdr_br ram_bw;              // MDR <-- RAM[mar]
    ioa_br mdr_bw;              // IOA <-- MDR
    //
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    mdr_br ram_bw;              // MDR <-- RAM[mar]
    a_br io_bw io_isack;        // A <-- I/O is ack
    a_br alu_f=not_a alu_bw fl_ar;  // A <-- NOT A
    a_br alu_f=not_a alu_bw fl_ar;  // A <-- NOT A
    pc_br sel_f=if_not_zero sel_bw  // PC = (not_zero ? MAR : PC)
    ctrl_start ctrl_load;       // CNT <-- 0
…
end
```

Figura u113.16. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

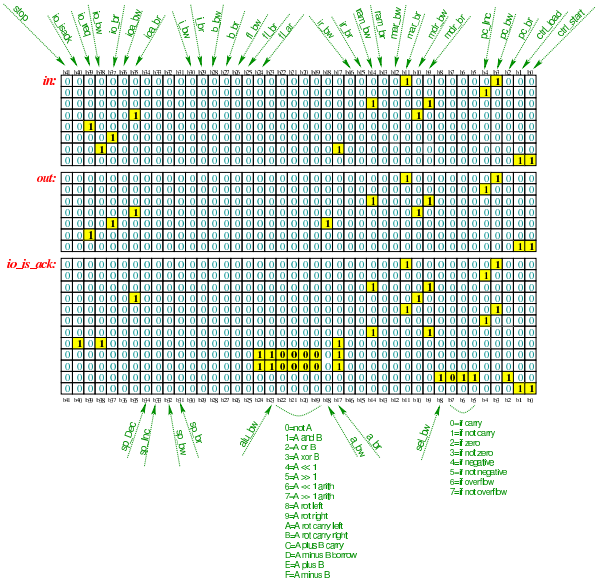


Tabella u113.17. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`in` ***indirizzo_io***	Legge un byte dal dispositivo a cui corrisponde l'indirizzo.
`out` ***indirizzo_io***	Scrive un byte sul dispositivo a cui corrisponde l'indirizzo.
`io_is_ack` ***indirizzo_io indirizzo***	Legge dal dispositivo indicato dal primo argomento, un codice che consente di verificare il ricevimento della conferma (*acknowledge*); se l'esito è valido, salta all'indirizzo indicato come secondo argomento.

Istruzione «out»

In questa sezione viene mostrato l'uso della macroistruzione '**`out`**', per visualizzare un carattere attraverso il dispositivo **`DSP`**. Nel listato successivo si mostra l'uso di '**`out`**' e poi anche '**`io_is_ack`**' per verificare che il carattere da visualizzare sia stato effettivamente mostrato. Il programma si limita a mostrare la lettera «H», ripetutamente, senza fermarsi.

Listato u113.18. Macrocodice per sperimentare l'istruzione **`out`** e **`io_is_ack`**: si vuole visualizzare la lettera «H», ripetutamente, controllando ogni volta il completamento dell'operazione. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-h-out.gm* .

```
begin macrocode @ 0
start:
        load_imm #carattere
```

```
        move_mdr_a
        out 0                   // display
check_ack:
        io_is_ack 0, #start
        jump #check_ack
stop:
        stop
carattere:
        .byte 0x48              // 'H'
end
```

Anche per questo esempio è disponibile un video: *http://www.youtube.com/watch?v=S9XqmTMYAj4* .

Istruzione «in»

In questa sezione viene mostrato l'uso della macroistruzione '**in**', per recepire la digitazione da tastiera, attraverso il modulo **KBD**. Nel listato successivo si usa anche l'istruzione '**out**', usata per riemettere il carattere ricevuto.

Listato u113.19. Macrocodice per sperimentare l'istruzione **in**: si vuole recepire la digitazione da tastiera, la quale viene riemessa attraverso l'istruzione **out** sul dispositivo **DSP**. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-h-in.gm* .

```
begin macrocode @ 0
start:
        in 1
        not
        not
        jump_if_zero #start
        out 0
        jump #start
stop:
        stop
end
```

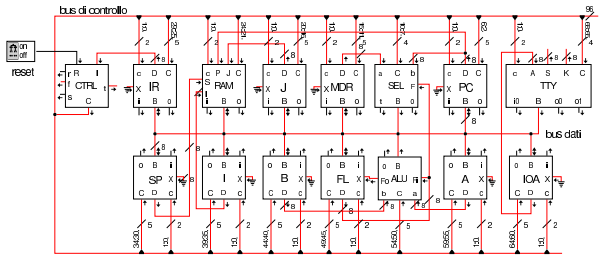
Video: *http://www.youtube.com/watch?v=JhGoQhssWQM* .

Versione I: ottimizzazione

Registri uniformi 885
RAM 886
Modulo «SEL» 888
ALU 888
Terminale 889
Unità di controllo 891
Memorie, campi, argomenti e codici operativi 893
Microcodice 899
Macrocodice: chiamata di una routine 907
Macrocodice: inserimento da tastiera e visualizzazione sullo schermo 907

Viene proposta una ristrutturazione della CPU dimostrativa sviluppata fino a questo punto, per riordinarne e semplificarne il funzionamento. Si parte dalla realizzazione uniforme dei registri, raccogliendo dove possibile le linee di controllo, per arrivare a un'ottimizzazione del funzionamento, evitando cicli di clock inutili.

Figura u114.1. CPU dimostrativa, versione «I».



Registri uniformi

I registri della nuova versione della CPU dimostrativa, hanno tutti la possibilità di incrementare o ridurre il valore che contengono, di una unità; inoltre, hanno la possibilità di leggere un dato dal bus (***B***) oppure da un ingresso ausiliario (***X***). Per poter monitorare la loro attività, dispongono di due uscite a cui si potrebbero collegare dei led, i quali si attivano in corrispondenza di una fase di lettura o di scrittura.

Figura u114.2. Aspetto esterno del registro generalizzato.

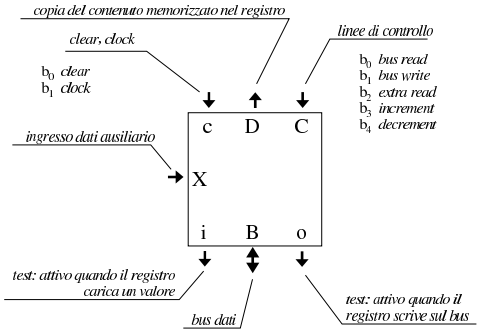
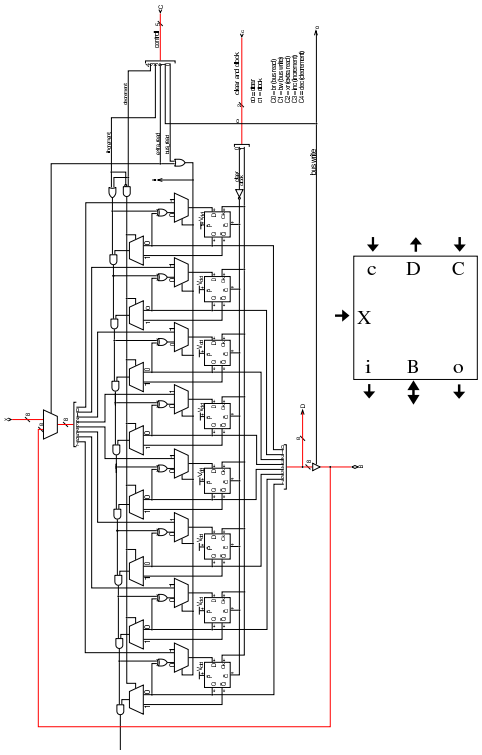


Figura u114.3. Schema interno del registro generalizzato.

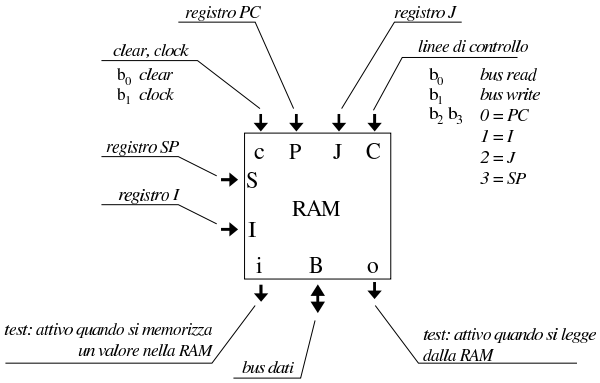


RAM

«

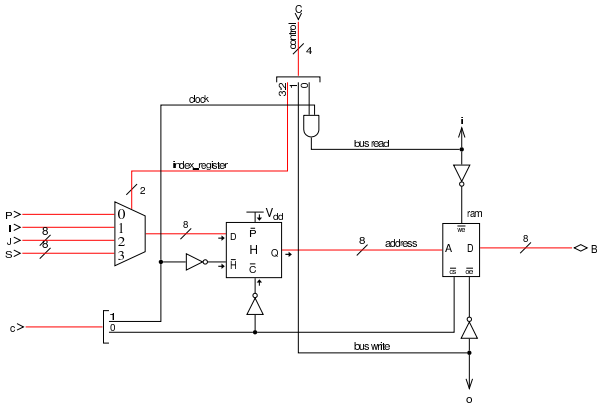
Il modulo **RAM** può ricevere l'indirizzo, direttamente dai registri ***PC***, ***SP***, ***I*** e ***J***, senza mediazioni; pertanto, il registro ***MAR*** utilizzato fino alla versione precedente è stato rimosso. La scelta del registro da cui leggere l'indirizzo dipende dal codice contenuto nel gruppo di linee dell'ingresso ***C***.

Figura u114.4. Aspetto esterno del modulo **RAM**.



Lo schema interno del modulo **RAM** cambia sostanzialmente, per consentire di utilizzare l'indirizzo proveniente dal registro selezionato, ma solo allo stato in cui questo dato risulta valido. Nello schema si vede l'aggiunta di un modulo, denominato ***H***, corrispondente a un registro controllato da un ingresso di abilitazione. Pertanto, tale registro non reagisce alla variazione dell'impulso di clock, ma si limita a mantenere memorizzato un valore per tutto il tempo in cui l'ingresso ***H'*** risulta azzerato.

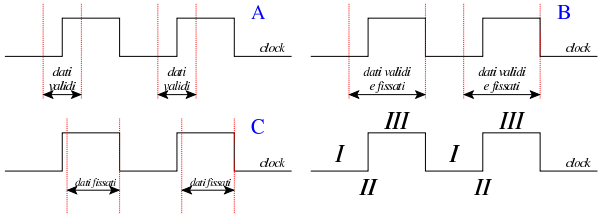
Figura u114.5. Schema interno del modulo **RAM**.



In questa versione della CPU, durante un ciclo di clock, l'indirizzo che serve a individuare la cella di memoria a cui si è interessati, può non essere stabile, a causa di vari fattori. Prima di tutto, l'indirizzo viene scelto attraverso un multiplatore, pescandolo da quattro registri diversi, ma questa selezione avviene all'inizio della fase «I» della figura successiva; pertanto, in questa prima fase l'informazione subisce un cambiamento nella maggior parte dei casi. Inoltre, quando scatta il segnale di clock, passando da zero a uno, il registro da cui si attinge l'informazione dell'indirizzo potrebbe essere indotto ad aggiornarsi, in preparazione della fase successiva. Quindi, l'informazione valida sull'indirizzo da utilizzare per la memoria RAM appare a cavallo della variazione positiva del segnale di clock (fase «II»). Tuttavia, quando si richiede di scrivere nella RAM un valore, la RAM stessa ha bisogno di disporre dell'indirizzo per un certo tempo, durante il quale questo indirizzo non deve cambiare; pertanto, si utilizza il registro ***H*** che è trasparente quando il segnale di clock è a zero, mentre blocca il proprio valore quando il segnale di clock è attivo. Per questo, la RAM viene abilitata a ricevere la richiesta di lettura o di scrittura soltanto durante il periodo attivo del segnale di clock (fase «III»). Quando si tratta invece di leggere dalla RAM, è sufficiente che la RAM abbia avuto il tempo di fornire il dato corrispondente all'indirizzo selezionato, nel momento in cui l'informazione viene poi attinta dal bus da un altro registro.

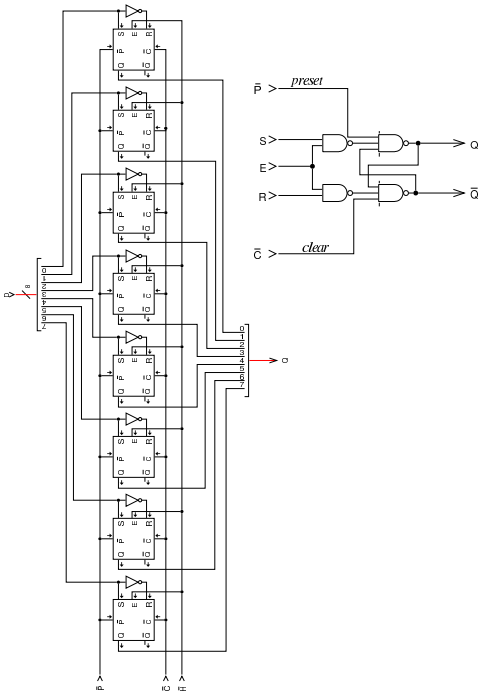
Nella figura, il grafico «A» si riferisce agli intervalli di validità dell'informazione degli indirizzi, a cavallo della variazione positiva del segnale di clock. Il grafico «B» mostra la situazione all'uscita del registro ***H***, che estende la validità dell'indirizzo ricevuto, in ingresso, perché quando il segnale di clock diventa positivo, blocca il valore alla sua uscita. Il grafico «C» mostra il periodo in cui è concesso alla memoria RAM di operare per modificare il proprio contenuto.

Figura u114.6. Fasi nel funzionamento del modulo **RAM**.



Il registro ***H*** è fatto di flip-flop SR semplici, collegati in modo da operare in qualità di flip-flop D, con ingresso di abilitazione. L'uso di flip-flop semplici, in questo caso, serve a evitare di introdurre latenze eccessive.

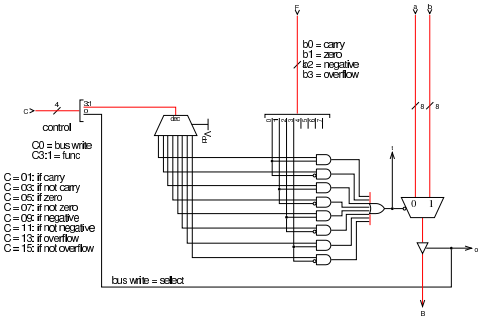
Figura u114.7. Schema interno del registro ***H*** (*hold*), contenuto del modulo **RAM**.



Modulo «SEL»

Il modulo di selezione non è cambiato, a parte la riorganizzazione del cablaggio e l'aggiunta di un'uscita diagnostica per sapere quando la condizione sottoposta a valutazione risulta avverarsi (uscita ***t***).

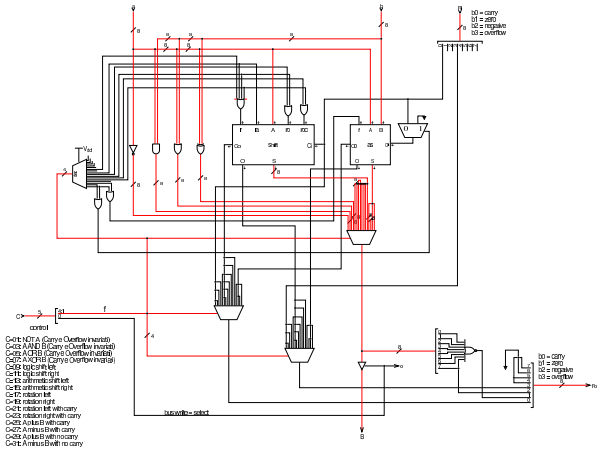
Figura u114.8. Schema interno del modulo **SEL**.



ALU

Anche la ALU non ha subito cambiamenti, a parte il fatto di avere riunito le linee di controllo e di disporre di un indicatore (uscita ***o***) che si attiva quando la ALU scrive sul bus dati un valore.

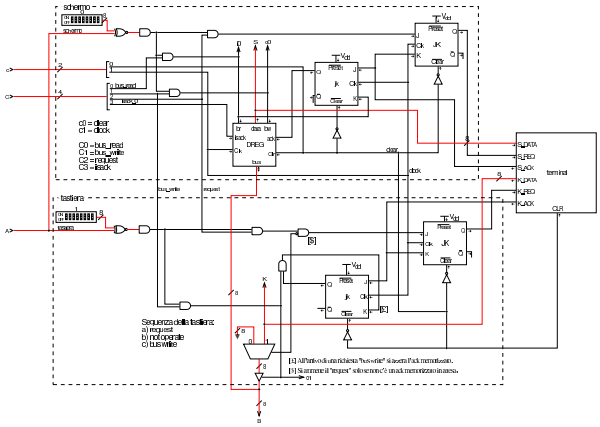
Figura u114.9. Schema interno del modulo **ALU**.



Terminale

Il terminale, costituito dal complesso tastiera-schermo, cambia rispetto alla versione precedente della CPU dimostrativa, in quanto torna a unificarsi, così come è realizzato nella versione già disponibile per Tkgate. Tuttavia, l'unificazione mantiene internamente la distinzione circuitale della versione precedente e anche la stessa logica di funzionamento; in pratica, si gestiscono sempre tastiera e schermo separatamente, ma nella realizzazione del codice TCL/Tk, si ha un modulo unico, che si manifesta così in una sola finestra durante la simulazione di Tkgate.

Figura u114.10. Circuito interno del modulo **TTY**: il registro ***DREG*** è identico a quello usato nella versione precedente.



Nella figura che mostra il circuito del modulo **TTY**, si può osservare la delimitazione tra le due porzioni, relative a tastiera e schermo: va notato che i due blocchi sono attivati attraverso indirizzi diversi (ingresso ***A***), esattamente come nella versione precedente. Il modulo **terminal** è scritto in Verilog, come già fatto nella versione precedente, solo che in questo caso si tratta di un modulo unico, per tastiera e schermo. A sua volta, il modulo **terminal** si avvale di codice TCL/Tk, costituito dal file '`terminal.tcl`' che viene mostrato subito dopo.

Listato u114.11. Modulo **terminal**, scritto in Verilog.

```
module terminal(K_DATA, K_REQ, K_ACK, S_DATA, S_REQ, S_ACK, CLR);

output K_ACK;
output S_ACK;
output [7:0] K_DATA;
input [7:0] S_DATA;
input K_REQ;
input S_REQ;
input CLR;
reg k_ready;
reg [7:0] key;
```

```
reg s_ready;

  initial
    begin
      k_ready = 0;
      s_ready = 0;
      key = 0;
    end

  always
    begin
      @(posedge CLR)
      k_ready = 0;
      s_ready = 0;
      key = 0;
    end

  initial $tkg$post("TERMINAL","%m");

  always
    begin
      @ (posedge K_REQ);
      # 5;
      key = $tkg$recv("%m.KD");
      # 5;
      k_ready = 1'b1;
      # 5;
      @ (negedge K_REQ);
      # 5;
      k_ready = 1'b0;
    end

  always
    begin
      @(posedge S_REQ);
      # 5;
      $tkg$send("%m.SD",S_DATA);
      # 5;
      s_ready = 1'b1;
      # 5;
      @(negedge S_REQ);
      # 5;
      s_ready = 1'b0;
    end

  assign S_ACK = s_ready;
  assign K_DATA = key;
  assign K_ACK = k_ready;

endmodule
```

Listato u114.12. File '`share/tkgate/vpd/terminal.tcl`'. Il file è molto simile a quello fornito assieme a Tkgate, per la gestione di un terminale.

```
image create bitmap txtcurs -file "$bd/txtcurs.b"

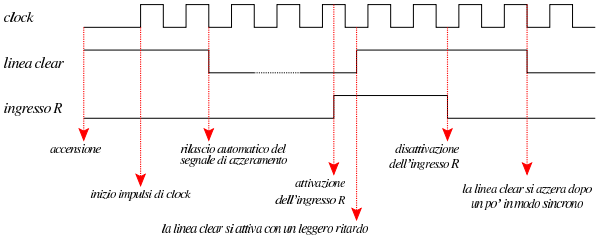
VPD::register TERMINAL
VPD::allow TERMINAL::post
VPD::allow TERMINAL::data

namespace eval TERMINAL {
    # Dichiarazione delle variabili pubbliche: le variabili
    # $terminal_... sono array dei quali si utilizza solo
    # l'elemento $n, il quale identifica univocamente l'istanza
    # dell'interfaccia in funzione.
    variable terminal_w
    variable terminal_pos
    #
    variable KD
    # Funzione richiesta da Tkgate per creare l'interfaccia.
    proc post {n} {
        variable terminal_w
        variable terminal_pos
        # Crea una finestra e salva l'oggetto in un elemento dell'array
        # $terminal_w.
        set terminal_w($n) [VPD::createWindow "TERMINAL $n" -shutdowncommand "TERMINAL::unpost $n"]
        # Per maggiore comodità, copia il riferimento all'oggetto nella
        # variabile locale $w e in seguito fa riferimento all'oggetto
        # attraverso questa seconda variabile.
        set w $terminal_w($n)
        text $w.txt -state disabled
        pack $w.txt
        # Mette il cursore alla fine del testo visualizzato.
        $w.txt image create end -image txtcurs
        # Collega la digitazione della tastiera, relativa all'oggetto
        # rappresentato da $terminal_w($n), alla funzione sendChar.
        bind $w <KeyPress> "TERMINAL::sendChar $n \"%A\""
        # Apre un canale di lettura, denominato «SD» (screen data),
        # associandolo alla funzione «data»; inoltre, apre un canale
        # di scrittura, denominato «KD» (keyboard data).
        if {[info exists ::tkgate_isInitialized]} {
            VPD::outsignal $n.KD TERMINAL::KD($n)
            VPD::insignal $n.SD -command "TERMINAL::data $n" -format %d
        }
        # Azzera il contatore che tiene conto dei caratteri visualizzati
        # sullo schermo.
        set terminal_pos($n) 0
    }
    # Funzione che recepisce la digitazione e la immette nel canale
    # denominato «KD», relativo all'istanza attuale dell'interfaccia.
    proc sendChar {n key} {
        variable KD
        if { [string length $key ] == 1 } {
            binary scan $key c c
            set TERMINAL::KD($n) $c
        }
    }
    # Funzione richiesta da Tkgate per distruggere l'interfaccia.
    proc unpost {n} {
        variable terminal_w
        variable terminal_pos
        destroy $terminal_w($n)
        destroy $terminal_pos($n)
        unset terminal_w($n)
        unset terminal_pos($n)
    }
    # Funzione usata per recepire i dati da visualizzare.
    proc data {n c} {
        variable terminal_w
        variable terminal_pos
        # Per maggiore comodità, copia il riferimento all'oggetto che
        # rappresenta l'interfaccia nella variabile $w.
        set w $terminal_w($n)
        catch {
            # La variabile $c contiene il carattere da visualizzare.
            if { $c == 7 } {
                # BEL
                bell
                return
            } elseif { $c == 127 || $c == 8 } {
                # DEL | BS
                if { $terminal_pos($n) > 0 } {
                    # Cancella l'ultimo carattere visualizzato, ma solo
                    # se il contatore dei caratteri è maggiore di zero,
                    # altrimenti sparirebbe il cursore e la
                    # visualizzazione verrebbe collocata in un'area
                    # non visibile dello schermo.
                    $w.txt configure -state normal
                    $w.txt delete "end - 3 chars"
                    $w.txt see end
                    $w.txt configure -state disabled
                    set terminal_pos($n) [expr {$terminal_pos($n) - 1}]
                }
                return
            } elseif { $c == 13 } {
                # CR viene trasformato in LF.
                set c 10
            }
            # Converte il numero del carattere in un simbolo
            # visualizzabile.
            set x [format %c $c]
            # Visualizza il simbolo.
            $w.txt configure -state normal
            $w.txt insert "end - 2 chars" $x
            $w.txt see end
            $w.txt configure -state disabled
            # Aggiorna il contatore dei caratteri visualizzati.
            set terminal_pos($n) [expr {$terminal_pos($n) + 1}]
        }
    }
}
```

Unità di controllo

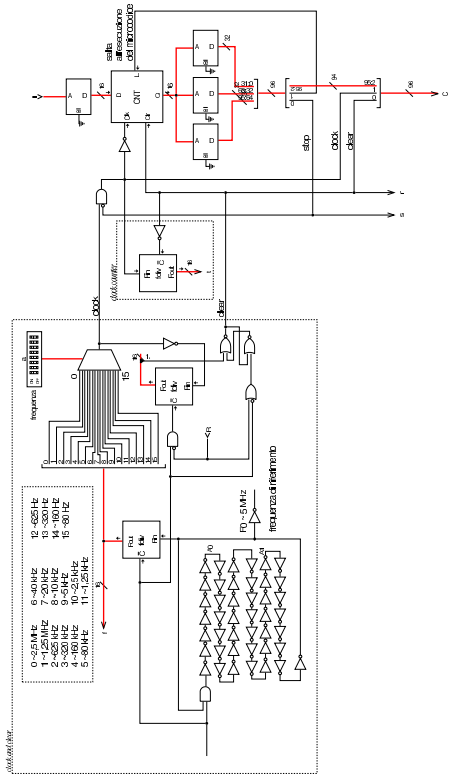
Per semplificare l'organizzazione del cablaggio, l'unità di controllo incorpora anche il generatore degli impulsi di clock; inoltre, il generatore di impulsi di clock incorpora la gestione del segnale di azzeramento, in modo che venga tolto solo nel momento più adatto rispetto all'impulso di clock: fino alla versione precedente della CPU dimostrativa, il circuito richiedeva un azzeramento manuale prima di poter iniziare a lavorare correttamente, inoltre il rilascio del segnale di azzeramento poteva avvenire in un momento inadatto che rendeva instabile il funzionamento.

Figura u114.13. Tempistica del funzionamento della linea ***clear***.



La figura successiva mostra lo schema dell'unità di controllo che integra le funzionalità di clock. Nella parte sinistra si trova il circuito che serve a generare gli impulsi di clock e a controllare la linea di azzeramento (*clear*). Va osservato che il modulo **`fdiv`** è esteso rispetto alla versione precedente, in modo da poter dividere la frequenza maggiormente; inoltre, la selezione della frequenza avviene attraverso un interruttore multiplo collegato a un multiplatore che si vede in alto. Tuttavia, dagli esperimenti fatti con Tkgate, la CPU funziona con una frequenza di clock non superiore a 1,25 MHz, pari al valore 1 per questo interruttore multiplo.

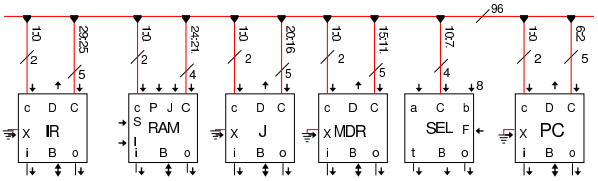
Figura u114.14. Schema completo dell'unità di controllo.



Nel circuito si usano diversi moduli **`fdiv`**: quello centrale serve a contare gli impulsi per sincronizzare la linea di azzeramento; quello più a destra serve a contare gli impulsi di clock a partire dall'avvio della CPU e consentirne il monitoraggio attraverso l'uscita ***t***: si tratta quindi soltanto di un ausilio diagnostico.

Nella parte destra che rappresenta l'unità di controllo originale, si vede un modulo contatore unico, a 16 bit (**`CNT`**), ma senza altre modifiche; inoltre, si vede che manca la possibilità di riportare l'esecuzione del microcodice all'indirizzo zero. Nelle linee che costituiscono assieme il bus di controllo, le prime due sono utilizzate per portare l'impulso di clock e il segnale di azzeramento (*clear*); le linee corrispondenti che escono dalla memoria che contiene il microcodice, servono per controllare l'unità stessa e non riguardano il resto della CPU. Allo stato attuale, questa versione dell'unità di controllo non permette di far riprendere il segnale di clock quando si attiva la linea interna di stop.

Figura u114.15. Connessione al bus di controllo.



Nella figura precedente si vedono alcuni componenti della CPU dimostrativa connessi al bus di controllo. Tutti questi componenti hanno in comune gli ingressi ***c*** (minuscola) e ***C*** (maiuscola). L'ingresso ***c*** è collegato sempre alle prime due linee del bus di controllo, dalle quali si ottiene, rispettivamente, il segnale di azzeramento e il segnale di clock. L'ingresso ***C***, invece, va connesso alle linee del bus di controllo che riguardano specificatamente il modulo. Nel caso dei registri uniformati, queste linee sono sempre cinque: *bus read*, *bus write*, *extra read*, *increment*, *decrement*. Il registro ***PC*** collega il proprio ingresso ***C*** alle linee da 2 a 6, del bus di controllo; il modulo **`SEL`** (che usa solo quattro linee di controllo) si collega alle linee da

7 a 10, e così si prosegue con gli altri componenti.

Memorie, campi, argomenti e codici operativi

Il sorgente Tkgate che serve a descrivere il contenuto delle memorie utilizzate con la CPU dimostrativa, inizia sempre con la definizione delle dimensioni di queste, assieme al loro nome:

```
map       bank[7:0]   ctrl.map;
microcode bank[31:0]  ctrl.micro0;
microcode bank[63:32] ctrl.micro1;
microcode bank[91:64] ctrl.micro2;
macrocode bank[7:0]   ram.ram;
```

In questa versione della CPU dimostrativa vengono cambiati leggermente i nomi delle memorie, in modo da rendere più chiaro il compito rispettivo. Va osservato che si utilizzano tre moduli di memoria, ognuno da 32 bit, per il microcodice, perché il bus di controllo prevede l'uso di molte linee.

```
field ctrl[1:0]  = {nop=0, stop=1, load=2};
field pc[6:2]    = {br=1, bw=2, xr=4, inc=8, dec=16};
field sel[10:7]  = {if_carry=1, if_not_carry=3,
                    if_zero=5, if_not_zero=7,
                    if_negative=9, if_not_negative=11,
                    if_overflow=13, if_not_overflow=15};
field mdr[15:11] = {br=1, bw=2, xr=4, inc=8, dec=16};
field j[20:16]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field ram[24:21] = {br=1, bw=2, p=0, i=4, j=8, s=12};
field ir[29:25]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field sp[34:30]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field i[39:35]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field b[44:40]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field fl[49:45]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field alu[54:50] = {not=1, and=3, or=5, xor=7, lshl=9, lshr=11,
                    ashl=13, ashr=15, rotl=17, rotr=19, rotcl=21,
                    rotcr=23, add_c=25, sub_b=27, add=29, sub=31};
field a[59:55]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field ioa[64:60] = {br=1, bw=2, xr=4, inc=8, dec=16};
field ioc[68:65] = {br=1, bw=2, req=4, isack=8};
```

I campi delle linee di controllo sono scritti in modo più compatto. Va osservato che i valori rappresentabili in ogni campo possono sommarsi con l'operatore OR binario. In pratica, in relazione al campo ***pc***, il quale si riferisce alle linee di controllo specifiche del registro ***PC***, è possibile attivare sia la scrittura sul bus dati (**`pc=bw`**), sia incrementare il valore del registro (**`pc=inc`**), nello stesso ciclo di clock.

Nella dichiarazione della memoria si vede che le prime due linee sono relative all'unità di controllo, ma va ricordato che poi quelle due linee non vengono convogliate al bus di controllo esterno, perché al loro posto si fa transitare la linea di azzeramento e quella di clock.

In questa versione esiste la possibilità di dichiarare una microistruzione nulla, al solo scopo di far passare un ciclo di clock, indicando **`ctrl=nop`**.

Il codice operativo delle istruzioni rimane a 8 bit, poi ci possono essere un massimo di due argomenti (da 8 bit ognuno):

```
operands op_0 {
    -  = { };
};
operands op_1 {
    #1 = { +1=#1[7:0]; };
};
operands op_2 {
    #1,#2 = { +1=#1[7:0]; +2=#2[15:8]; };
};
```

Il codice operativo delle istruzioni disponibili è semplicemente un numero intero che parte da zero con l'istruzione **`nop`** e arriva a 255 con l'istruzione **`stop`**, senza altri accorgimenti:

```
op nop {                  // not operate
  map nop: 0;
  +0[7:0]=0;
  operands op_0;
};
op load                   // MDR <-- RAM[arg]
{
  map load: 1;
```

```
  +0[7:0]=1;
  operands op_1;
};
op load_i               // MDR <-- RAM[i]
{
  map load_i: 2;
  +0[7:0]=2;
  operands op_0;
};
op load_j               // MDR <-- RAM[j]
{
  map load_j: 3;
  +0[7:0]=3;
  operands op_0;
};
op store {              // RAM[arg] <-- MDR
  map store: 4;
  +0[7:0]=4;
  operands op_1;
};
op store_i {            // RAM[i] <-- MDR
  map store_i: 5;
  +0[7:0]=5;
  operands op_0;
};
op store_j {            // RAM[j] <-- MDR
  map store_j: 6;
  +0[7:0]=6;
  operands op_0;
};
op cp_ij {              // RAM[j++] <-- MDR <-- RAM[i++]
  map cp_ij: 7;
  +0[7:0]=7;
  operands op_0;
};
op cp_ji {              // RAM[i++] <-- MDR <-- RAM[j++]
  map cp_ji: 8;
  +0[7:0]=8;
  operands op_0;
};
op mv_mdr_a {           // A <-- MDR
  map mv_mdr_a: 9;
  +0[7:0]=9;
  operands op_0;
};
op mv_mdr_b {           // B <-- MDR
  map mv_mdr_b: 10;
  +0[7:0]=10;
  operands op_0;
};
op mv_mdr_fl {          // FL <-- MDR
  map mv_mdr_fl: 11;
  +0[7:0]=11;
  operands op_0;
};
op mv_mdr_sp {          // SP <-- MDR
  map mv_mdr_sp: 12;
  +0[7:0]=12;
  operands op_0;
};
op mv_mdr_i {           // I <-- MDR
  map mv_mdr_i: 13;
  +0[7:0]=13;
  operands op_0;
};
op mv_mdr_j {           // J <-- MDR
  map mv_mdr_j: 14;
  +0[7:0]=14;
  operands op_0;
};
op mv_a_mdr {           // A <-- MDR
  map mv_a_mdr: 15;
  +0[7:0]=15;
  operands op_0;
};
op mv_a_b {             // B <-- A
  map mv_a_b: 16;
  +0[7:0]=16;
  operands op_0;
};
op mv_a_fl {            // FL <-- A
```



```
  map mv_a_fl: 17;
  +0[7:0]=17;
  operands op_0;
};
op mv_a_sp {            // SP <-- A
  map mv_a_sp: 18;
  +0[7:0]=18;
  operands op_0;
};
op mv_a_i {             // I <-- A
  map mv_a_i: 19;
  +0[7:0]=19;
  operands op_0;
};
op mv_a_j {             // J <-- A
  map mv_a_j: 20;
  +0[7:0]=20;
  operands op_0;
};
op mv_b_a {              // A <-- B
  map mv_b_a: 21;
  +0[7:0]=21;
  operands op_0;
};
op mv_b_mdr {           // MDR <-- B
  map mv_b_mdr: 22;
  +0[7:0]=22;
  operands op_0;
};
op mv_b_fl {            // FL <-- B
  map mv_b_fl: 23;
  +0[7:0]=23;
  operands op_0;
};
op mv_b_sp {            // SP <-- B
  map mv_b_sp: 24;
  +0[7:0]=24;
  operands op_0;
};
op mv_b_i {             // I <-- B
  map mv_b_i: 25;
  +0[7:0]=25;
  operands op_0;
};
op mv_b_j {             // J <-- B
  map mv_b_j: 26;
  +0[7:0]=26;
  operands op_0;
};
op mv_fl_a {            // A <-- FL
  map mv_fl_a: 27;
  +0[7:0]=27;
  operands op_0;
};
op mv_fl_b {            // B <-- FL
  map mv_fl_b: 28;
  +0[7:0]=28;
  operands op_0;
};
op mv_fl_mdr {          // MDR <-- FL
  map mv_fl_mdr: 29;
  +0[7:0]=29;
  operands op_0;
};
op mv_fl_sp {           // SP <-- FL
  map mv_fl_sp: 30;
  +0[7:0]=30;
  operands op_0;
};
op mv_fl_i {            // I <-- FL
  map mv_fl_i: 31;
  +0[7:0]=31;
  operands op_0;
};
op mv_fl_j {            // J <-- FL
  map mv_fl_j: 32;
  +0[7:0]=32;
  operands op_0;
};
op mv_sp_a {            // A <-- SP
  map mv_sp_a: 33;
```

```
  +0[7:0]=33;
  operands op_0;
};
op mv_sp_b {            // B <-- SP
  map mv_sp_b: 34;
  +0[7:0]=34;
  operands op_0;
};
op mv_sp_fl {           // FL <-- SP
  map mv_sp_fl: 35;
  +0[7:0]=35;
  operands op_0;
};
op mv_sp_mdr {          // MDR <-- SP
  map mv_sp_mdr: 36;
  +0[7:0]=36;
  operands op_0;
};
op mv_sp_i {            // I <-- SP
  map mv_sp_i: 37;
  +0[7:0]=37;
  operands op_0;
};
op mv_sp_j {            // J <-- SP
  map mv_sp_j: 38;
  +0[7:0]=38;
  operands op_0;
};
op mv_i_a {             // A <-- I
  map mv_i_a: 39;
  +0[7:0]=39;
  operands op_0;
};
op mv_i_b {             // B <-- I
  map mv_i_b: 40;
  +0[7:0]=40;
  operands op_0;
};
op mv_i_fl {            // FL <-- I
  map mv_i_fl: 41;
  +0[7:0]=41;
  operands op_0;
};
op mv_i_sp {            // SP <-- I
  map mv_i_sp: 42;
  +0[7:0]=42;
  operands op_0;
};
op mv_i_mdr {           // MDR <-- I
  map mv_i_mdr: 43;
  +0[7:0]=43;
  operands op_0;
};
op mv_i_j {             // J <-- I
  map mv_i_j: 44;
  +0[7:0]=44;
  operands op_0;
};
op mv_j_a {             // A <-- J
  map mv_j_a: 45;
  +0[7:0]=45;
  operands op_0;
};
op mv_j_b {             // B <-- J
  map mv_j_b: 46;
  +0[7:0]=46;
  operands op_0;
};
op mv_j_fl {            // FL <-- J
  map mv_j_fl: 47;
  +0[7:0]=47;
  operands op_0;
};
op mv_j_sp {            // SP <-- J
  map mv_j_sp: 48;
  +0[7:0]=48;
  operands op_0;
};
op mv_j_i {             // I <-- J
  map mv_j_i: 49;
  +0[7:0]=49;
```



```
  operands op_0;
};
op mv_j_mdr {           // MDR <-- J
  map mv_j_mdr: 50;
  +0[7:0]=50;
  operands op_0;
};
op jump {               // PC <-- arg
  map jump: 51;
  +0[7:0]=51;
  operands op_1;
};
op jump_c {             // if carry, PC <-- arg
  map jump_c: 52;
  +0[7:0]=52;
  operands op_1;
};
op jump_nc {            // if not carry, PC <-- arg
  map jump_nc: 53;
  +0[7:0]=53;
  operands op_1;
};
op jump_z {             // if zero, PC <-- arg
  map jump_z: 54;
  +0[7:0]=54;
  operands op_1;
};
op jump_nz {            // if not zero, PC <-- arg
  map jump_nz: 55;
  +0[7:0]=55;
  operands op_1;
};
op jump_n {             // if negative, PC <-- arg
  map jump_n: 56;
  +0[7:0]=56;
  operands op_1;
};
op jump_nn {            // if not negative, PC <-- arg
  map jump_nn: 57;
  +0[7:0]=57;
  operands op_1;
};
op jump_o {             // if overflow, PC <-- arg
  map jump_o: 58;
  +0[7:0]=58;
  operands op_1;
};
op jump_no {            // if not overflow, PC <-- arg
  map jump_no: 59;
  +0[7:0]=59;
  operands op_1;
};
op call {
  map call : 60;
  +0[7:0]=60;
  operands op_1;
};
op call_i {             // call I
  map call_i: 61;
  +0[7:0]=61;
  operands op_0;
};
op call_j {             // call J
  map call_j: 62;
  +0[7:0]=62;
  operands op_0;
};
op return {
  map return : 63;
  +0[7:0]=63;
  operands op_0;
};
op push_mdr {
  map push_mdr: 64;
  +0[7:0]=64;
  operands op_0;
};
op push_a {
  map push_a: 65;
  +0[7:0]=65;
  operands op_0;
```

```
};
op push_b {
  map push_b: 66;
  +0[7:0]=66;
  operands op_0;
};
op push_fl {
  map push_fl: 67;
  +0[7:0]=67;
  operands op_0;
};
op push_i {
  map push_i: 68;
  +0[7:0]=68;
  operands op_0;
};
op push_j {
  map push_j: 69;
  +0[7:0]=69;
  operands op_0;
};
op pop_mdr {
  map pop_mdr: 70;
  +0[7:0]=70;
  operands op_0;
};
op pop_a {
  map pop_a: 71;
  +0[7:0]=71;
  operands op_0;
};
op pop_b {
  map pop_b: 72;
  +0[7:0]=72;
  operands op_0;
};
op pop_fl {
  map pop_fl: 73;
  +0[7:0]=73;
  operands op_0;
};
op pop_i {
  map pop_i: 74;
  +0[7:0]=74;
  operands op_0;
};
op pop_j {
  map pop_j: 75;
  +0[7:0]=75;
  operands op_0;
};
op not {
  map not: 76;
  +0[7:0]=76;
  operands op_0;
};
op and {
  map and: 77;
  +0[7:0]=77;
  operands op_0;
};
op or {
  map or: 78;
  +0[7:0]=78;
  operands op_0;
};
op xor {
  map xor: 79;
  +0[7:0]=79;
  operands op_0;
};
op lshl {
  map lshl: 80;
  +0[7:0]=80;
  operands op_0;
};
op lshr {
  map lshr: 81;
  +0[7:0]=81;
  operands op_0;
};
```



```
op ashl {
  map ashl: 82;
  +0[7:0]=82;
  operands op_0;
};
op ashr {
  map ashr: 83;
  +0[7:0]=83;
  operands op_0;
};
op rotl {
  map rotl: 84;
  +0[7:0]=84;
  operands op_0;
};
op rotr {
  map rotr: 85;
  +0[7:0]=85;
  operands op_0;
};
op rotcl {
  map rotcl: 86;
  +0[7:0]=86;
  operands op_0;
};
op rotcr {
  map rotcr: 87;
  +0[7:0]=87;
  operands op_0;
};
op add_c {
  map add_c: 88;
  +0[7:0]=88;
  operands op_0;
};
op sub_b {
  map sub_b: 89;
  +0[7:0]=89;
  operands op_0;
};
op add {
  map add: 90;
  +0[7:0]=90;
  operands op_0;
};
op sub {
  map sub: 91;
  +0[7:0]=91;
  operands op_0;
};
op in {
  map in : 92;
  +0[7:0]=92;
  operands op_1;
};
op out {
  map out: 93;
  +0[7:0]=93;
  operands op_1;
};
op is_ack {
  map is_ack: 94;
  +0[7:0]=94;
  operands op_2;
};
op stop {
  map stop : 255;
  +0[7:0]=255;
  operands op_0;
};
```

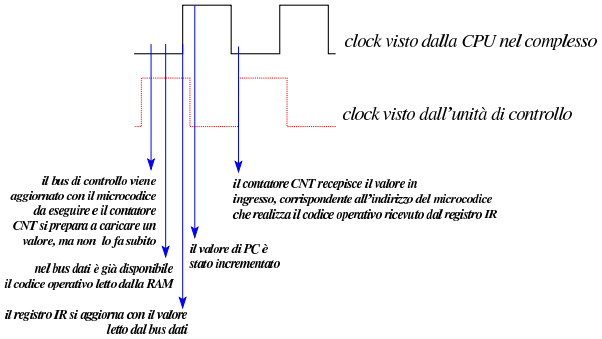
Microcodice

Nella descrizione del microcodice vero e proprio, si inizia con ciò che serve al caricamento del primo codice operativo dalla memoria, ma in questa realizzazione può avvenire tutto in un solo ciclo di clock:

```
begin microcode @ 0
//
    ir=br ram=bw ram=p pc=inc ctrl=load;   // IR <- RAM[pc++],
                                           // jump MAP[ir];
```

In pratica, il registro ***IR*** carica dal bus dati quanto emesso dal modulo **RAM**, il quale a sua volta riceve l'indirizzo dal registro ***PC***, il quale viene incrementato contestualmente. Oltre a questo, si richiede all'unità di controllo di aggiornare il proprio contatore con il valore proveniente dalla memoria ***ctrl.map*** in corrispondenza dell'indirizzo che rappresenta il codice operativo. Si può fare tutto questo in un solo ciclo di clock perché la struttura della CPU è cambiata rispetto alla versione precedente. Vanno considerate le diverse fasi del ciclo di clock, che intervengono in modo differente nell'unità di controllo rispetto ai componenti che poi sono connessi al bus di controllo, come si vede nella figura successiva.

Figura u114.21. Il ciclo di clock dell'operazione di caricamento e messa in esecuzione di un'istruzione contenuta nella memoria RAM.



Pertanto, con un solo ciclo di clock si realizza quello che è noto come *fetch*. Nella descrizione successiva delle istruzioni, alla fine di ogni procedimento, si ripete la microistruzione di *fetch*, senza bisogno di far ripartire il contatore ***CNT*** dalla prima microistruzione, come necessario, invece, nelle versioni precedenti della CPU dimostrativa.

Per la macroistruzione **nop**, in pratica, c'è solo la microistruzione di *fetch*:

```
nop:
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
```

Segue la descrizione delle altre macroistruzioni:

```
load:
    i=br ram=bw ram=p pc=inc;                   // I <- RAM[pc++];
    mdr=br ram=bw ram=i;                        // MDR <- RAM[i];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
load_i:
    mdr=br ram=bw ram=i;                        // MDR <- RAM[i];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
load_j:
    mdr=br ram=bw ram=j;                        // MDR <- RAM[j];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
store:
    i=br ram=bw ram=p pc=inc;                   // I <- RAM[pc++];
    ram=br ram=i mdr=bw;                        // RAM[i] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
store_i:
    ram=br ram=i mdr=bw;                        // RAM[i] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
store_j:
    ram=br ram=j mdr=bw;                        // RAM[j] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
cp_ij:
    mdr=br ram=bw ram=i i=inc;                  // MDR <- RAM[i++];
    ram=br ram=j mdr=bw j=inc;                  // RAM[j++] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
cp_ji:
    mdr=br ram=bw ram=j j=inc;                  // MDR <- RAM[j++];
    ram=br ram=i mdr=bw i=inc;                  // RAM[i++] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_a:
    a=br mdr=bw;                                // A <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_b:
    b=br mdr=bw;                                // B <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_fl:
    fl=br mdr=bw;                               // FL <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_sp:
    sp=br mdr=bw;                               // SP <- MDR;
```



```
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_i:
    i=br mdr=bw;                                // I <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_mdr_j:
    j=br mdr=bw;                                // J <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_mdr:
    mdr=br a=bw;                                // MDR <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_b:
    b=br a=bw;                                  // B <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_fl:
    fl=br a=bw;                                 // FL <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_sp:
    sp=br a=bw;                                 // SP <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_i:
    i=br  a=bw;                                 // I <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_a_j:
    j=br  a=bw;                                 // J <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_a:
    a=br b=bw;                                  // A <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_mdr:
    mdr=br b=bw;                                // MDR <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_fl:
    fl=br b=bw;                                 // FL <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_sp:
    sp=br b=bw;                                 // SP <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_i:
    i=br b=bw;                                  // I <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_b_j:
    j=br b=bw;                                  // J <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_a:
    a=br fl=bw;                                 // A <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_b:
    b=br fl=bw;                                 // B <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_mdr:
    mdr=br fl=bw;                               // MDR <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_sp:
    sp=br fl=bw;                                // SP <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_i:
    i=br fl=bw;                                 // I <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_fl_j:
    j=br fl=bw;                                 // J <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_a:
    a=br sp=bw;                                 // A <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_b:
    b=br sp=bw;                                 // B <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_fl:
    fl=br sp=bw;                                // FL <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_mdr:
    mdr=br sp=bw;                               // MDR <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_i:
    i=br sp=bw;                                 // I <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_sp_j:
    j=br sp=bw;                                 // J <- SP;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_a:
    a=br i=bw;                                  // A <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_b:
    b=br i=bw;                                  // B <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_fl:
    fl=br i=bw;                                 // FL <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_sp:
    sp=br i=bw;                                 // SP <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_mdr:
    mdr=br i=bw;                                // MDR <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_i_j:
    j=br i=bw;                                  // J <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_j_a:
    a=br j=bw;                                  // A <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
```

```
mv_j_b:
    b=br j=bw;                                  // B <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_j_fl:
    fl=br j=bw;                                 // FL <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_j_sp:
    sp=br j=bw;                                 // SP <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_j_i:
    i=br j=bw;                                  // I <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
mv_j_mdr:
    mdr=br j=bw;                                // MDR <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump:
    i=br pc=bw;                                 // I <- PC
    pc=br ram=bw ram=i;                         // PC <-- RAM[i]
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_c:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_carry;                         // PC = (carry?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_nc:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_not_carry;                     // PC = (not_carry?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_z:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_zero;                          // PC = (zero?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_nz:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_not_zero;                      // PC = (not_zero?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_n:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_negative;                      // PC = (negative?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_nn:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_not_negative;                  // PC = (not_negative?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_o:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_overflow;                      // PC = (overflow?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
jump_no:
    mdr=br ram=bw ram=p pc=inc;                 // MDR <-- RAM[pc++]
    pc=br sel=if_not_overflow;                  // PC = (not_overflow?MDR:PC)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
call:
    i=br ram=bw ram=p pc=inc sp=dec;            // I <- RAM[pc++], SP--;
    ram=br ram=s pc=bw;                         // RAM[sp] <- PC;
    pc=br i=bw;                                 // PC <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
call_i:
    sp=dec;                                     // SP--;
    ram=br ram=s pc=bw;                         // RAM[sp] <- PC;
    pc=br i=bw;                                 // PC <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
call_j:
    sp=dec;                                     // SP--;
    ram=br ram=s pc=bw;                         // RAM[sp] <- PC;
    pc=br j=bw;                                 // PC <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
return:
    pc=br ram=bw ram=s sp=inc;                  // PC <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_mdr:
    sp=dec;                                     // SP--;
    ram=br ram=s mdr=bw;                        // RAM[sp] <- MDR;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_a:
    sp=dec;                                     // SP--;
    ram=br ram=s a=bw;                          // RAM[sp] <- A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_b:
    sp=dec;                                     // SP--;
    ram=br ram=s b=bw;                          // RAM[sp] <- B;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_fl:
    sp=dec;                                     // SP--;
    ram=br ram=s fl=bw;                         // RAM[sp] <- FL;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_i:
    sp=dec;                                     // SP--;
    ram=br ram=s i=bw;                          // RAM[sp] <- I;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
push_j:
    sp=dec;                                     // SP--;
    ram=br ram=s j=bw;                          // RAM[sp] <- J;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_mdr:
    mdr=br ram=bw ram=s sp=inc;                 // MDR <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_a:
    a=br ram=bw ram=s sp=inc;                   // A <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_b:
```



```
    b=br ram=bw ram=s sp=inc;                   // B <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_fl:
    fl=br ram=bw ram=s sp=inc;                  // FL <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_i:
    i=br ram=bw ram=s sp=inc;                   // I <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
pop_j:
    j=br ram=bw ram=s sp=inc;                   // J <- RAM[sp++];
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
not:
    a=br alu=not fl=xr;                         // A <- NOT A;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
and:
    a=br alu=and fl=xr;                         // A <- A AND B
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
or:
    a=br alu=or fl=xr;                          // A <- A OR B
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
xor:
    a=br alu=xor fl=xr;                         // A <- A XOR B
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
lshl:
    a=br alu=lshl fl=xr;                        // A <- A << 1
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
lshr:
    a=br alu=lshr fl=xr;                        // A <- A >> 1
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
ashl:
    a=br alu=ashl fl=xr;                        // A <- A*2
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
ashr:
    a=br alu=ashr fl=xr;                        // A <- A/2
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
rotl:
    a=br alu=rotl fl=xr;                        // A <- rotl(A)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
rotr:
    a=br alu=rotr fl=xr;                        // A <- rotr(A)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
rotcl:
    a=br alu=rotcl fl=xr;                       // A <- rotcl(A)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
rotcr:
    a=br alu=rotcr fl=xr;                       // A <- rotcr(A)
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
add_c:
    a=br alu=add_c fl=xr;                       // A <- A+B+carry
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
sub_b:
    a=br alu=sub_b fl=xr;                       // A <- A-B-borrow
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
add:
    a=br alu=add fl=xr;                         // A <- A+B
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
sub:
    a=br alu=sub fl=xr;                         // A <- A-B
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
in:
    ioa=br ram=bw ram=p pc=inc;                 // IOA <- RAM[pc++];
    ioc=req;                                    // I/O request;
    ctrl=nop;                                   // non fa alcunché
    a=br ioc=bw;                                // A <- I/O
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
out:
    ioa=br ram=bw ram=p pc=inc;                 // IOA <- RAM[pc++];
    ioc=br a=bw;                                // I/O <- A
    ioc=req;                                    // I/O request;
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
is_ack:
    ioa=br ram=bw ram=p pc=inc;                 // IOA <- RAM[pc++];
    mdr=br ram=bw ram=p pc=inc;                 // MDR <- RAM[pc++];
    a=br ioc=bw ioc=isack;                      // A <- I/O is ack;
    a=br alu=not fl=xr;                         // A <- NOT A;
    a=br alu=not fl=xr;                         // A <- NOT A;
    pc=br sel=if_not_zero;                      // PC = (not_zero?MDR:PC);
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
stop:
    ctrl=stop;                                  // stop clock
    // if resumed:
    ir=br ram=bw ram=p pc=inc ctrl=load;        // fetch
end
```

L'insieme delle macroistruzioni è cambiato leggermente ed è anche esteso, in considerazione delle modifiche apportate alla CPU, come descritto nella tabella successiva.

Tabella u114.24. Elenco completo delle macroistruzioni disponibili nella versione attuale della CPU dimostrativa.

Sintassi	Cicli di clock	Descrizione
`nop`	1	Non fa alcunché.

Sintassi	Cicli di clock	Descrizione
`load` ***indirizzo***	3	Carica nel registro ***I*** l'argomento e nel registro ***MDR*** il contenuto della memoria all'indirizzo specificato.
`load_i` `load_j`	2	Carica nel registro ***MDR*** il contenuto della memoria all'indirizzo specificato dal registro ***I*** o ***J*** rispettivamente.
`store` ***indirizzo***	3	Carica nel registro ***I*** l'argomento e scrive in memoria, in corrispondenza dell'indirizzo indicato, quanto contenuto nel registro ***MDR***.
`store_i` `store_j`	2	Scrive in memoria, in corrispondenza dell'indirizzo contenuto nel registro ***I*** o ***J*** rispettivamente, quanto contenuto nel registro ***MDR***.
`cp_ij` `cp_ji`	3	Nel primo caso, copia il contenuto della memoria RAM in corrispondenza dell'indirizzo contenuto nel registro ***I***, all'indirizzo rappresentato dal registro ***J***, incrementando successivamente i due registri; nel secondo caso, la copia avviene in senso inverso, ma sempre con incremento successivo degli indici.

`mv_mdr_a`
`mv_mdr_b`
`mv_mdr_fl`
`mv_mdr_sp`
`mv_mdr_i`
`mv_mdr_j`
	2	Copia il contenuto del registro ***MDR*** nel registro ***A***, ***B***, ***FL***, ***SP***, ***I*** o ***J***, rispettivamente.

`mv_a_mdr`
`mv_a_b`
`mv_a_fl`
`mv_a_sp`
`mv_a_i`
`mv_a_j`
	2	Copia il contenuto del registro ***A*** nel registro ***MDR***, ***B***, ***FL***, ***SP***, ***I*** o ***J***, rispettivamente.

`mv_b_a`
`mv_b_mdrb`
`mv_b_fl`
`mv_b_sp`
`mv_b_i`
`mv_b_j`
	2	Copia il contenuto del registro ***B*** nel registro ***A***, ***MDR***, ***FL***, ***SP***, ***I*** o ***J***, rispettivamente.

`mv_fl_a`
`mv_fl_b`
`mv_fl_mdr`
`mv_fl_sp`
`mv_fl_i`
`mv_fl_j`
	2	Copia il contenuto del registro ***FL*** nel registro ***A***, ***B***, ***MDR***, ***SP***, ***I*** o ***J***, rispettivamente.

Sintassi	Cicli di clock	Descrizione

`mv_sp_a`
`mv_sp_b`
`mv_sp_fl`
`mv_sp_mdr`
`mv_sp_i`
`mv_sp_j`
	2	Copia il contenuto del registro ***SP*** nel registro ***A***, ***B***, ***FL***, ***MDR***, ***I*** o ***J***, rispettivamente.

`mv_i_a`
`mv_i_b`
`mv_i_fl`
`mv_i_sp`
`mv_i_mdr`
`mv_i_j`
	2	Copia il contenuto del registro ***I*** nel registro ***A***, ***B***, ***FL***, ***SP***, ***MDR*** o ***J***, rispettivamente.

`mv_j_a`
`mv_j_b`
`mv_j_fl`
`mv_j_sp`
`mv_j_i`
`mv_j_mdr`
	2	Copia il contenuto del registro ***J*** nel registro ***A***, ***B***, ***FL***, ***SP***, ***I*** o ***MDR***, rispettivamente.

`jump` ***indirizzo***	3	Mette nel registro ***I*** l'argomento e poi salta all'esecuzione dell'istruzione che si trova all'indirizzo indicato.
`jump_c` ***indirizzo*** `jump_nc` ***indirizzo*** `jump_z` ***indirizzo*** `jump_nz` ***indirizzo*** `jump_n` ***indirizzo*** `jump_nn` ***indirizzo*** `jump_o` ***indirizzo*** `jump_no` ***indirizzo***	3	Mette nel registro ***MDR*** l'argomento e poi, se la condizione si avvera, salta all'esecuzione dell'istruzione che si trova all'indirizzo indicato. Le condizioni sono, nell'ordine: esistenza di un riporto, assenza di un riporto, valore a zero, valore diverso da zero, valore negativo, valore non negativo, straripamento, non straripamento.
`call` ***indirizzo***	4	Mette nel registro ***I*** l'argomento e riduce di una unità il valore del registro ***SP***; poi, in corrispondenza dell'indirizzo di memoria contenuto nel registro ***SP***, salva il valore contenuto nel registro ***PC***: in pratica salva nella pila il valore di ***PC***. Subito dopo, salta all'indirizzo memorizzato nel registro ***I***.
`call_i` `call_j`	4	Riduce di una unità il valore del registro ***SP***; poi, in corrispondenza dell'indirizzo di memoria contenuto nel registro ***SP***, salva il valore contenuto nel registro ***PC***: in pratica salva nella pila il valore di ***PC***. Subito dopo, salta all'indirizzo memorizzato nel registro ***I*** o ***j***, rispettivamente.

Sintassi	Cicli di clock	Descrizione
`return`	2	Legge il valore contenuto nella posizione di memoria indicato dal registro ***SP*** e lo mette nel registro ***PC***, quindi incrementa di una unità il registro ***SP***. In pratica, estrae dalla pila l'indirizzo di ritorno di una chiamata precedente.

`push_mdr`
`push_a`
`push_b`
`push_fl`
`push_sp`
`push_i`
`push_j`
	3	Riduce di una unità il valore del registro ***SP***; poi, in corrispondenza dell'indirizzo di memoria contenuto nel registro ***SP***, salva il valore contenuto nel registro ***MDR***, ***A***, ***B***, ***FL***, ***SP***, ***I*** o ***J***, rispettivamente.

`pop_mdr`
`pop_a`
`pop_b`
`pop_fl`
`pop_sp`
`pop_i`
`pop_j`
	3	Copia, rispettivamente nel registro ***MDR***, ***A***, ***B***, ***FL***, ***SP***, ***I*** o ***J***, il contenuto dell'area di memoria corrispondente all'indirizzo indicato dal registro ***SP***, incrementando poi ***SP*** di una unità.

`not`	2	Esegue l'operazione logica NOT ***A***, bit per bit, mettendo il risultato nello stesso registro ***A***.

`and`
`or`
`xor`
	2	Esegue, rispettivamente, l'operazione logica ***A*** AND ***B***, ***A*** OR ***B***, ***A*** XOR ***B***, mettendo il risultato nel registro ***A***.

`lshl` `lshr`	2	Esegue, rispettivamente, lo scorrimento logico a sinistra o a destra, del contenuto del registro ***A***.
`ashl` `ashr`	2	Esegue, rispettivamente, lo scorrimento aritmetico a sinistra o a destra, del contenuto del registro ***A***.
`rotl` `rotl`	2	Esegue, rispettivamente, la rotazione a sinistra o a destra, del contenuto del registro ***A***.
`rotcl` `rotcl`	2	Esegue, rispettivamente, la rotazione con riporto a sinistra o a destra, del contenuto del registro ***A*** e dell'indicatore di riporto.
`add_c` `sub_b`	2	Esegue, rispettivamente, la somma (***A***+***B***) o la sottrazione (***A***−***B***), utilizzando il riporto o la richiesta di prestito precedente, mettendo il risultato nel registro ***A***.
`add` `sub`	2	Esegue, rispettivamente, la somma (***A***+***B***) o la sottrazione (***A***−***B***), ignorando il riporto o la richiesta di prestito precedente, mettendo il risultato nel registro ***A***.

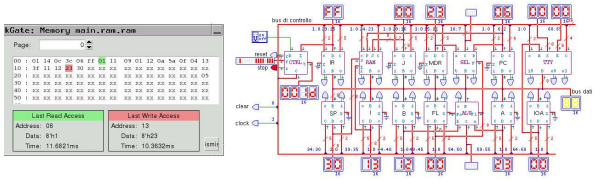
Sintassi	Cicli di clock	Descrizione
`in` ***indirizzo_io***	5	Legge un valore dal dispositivo di I/O individuato dall'indirizzo fornito, mettendo questo valore nel registro ***A***.
`out` ***indirizzo_io***	4	Scrive un valore sul dispositivo di I/O individuato dall'indirizzo fornito, utilizzando il valore contenuto nel registro ***A***.
`is_ack` ***indirizzo_io indirizzo***	7	Carica l'indirizzo rappresentato dal secondo argomento nel registro ***MDR*** e poi interroga un dispositivo di I/O per ottenere un codice di conferma da mettere nel registro ***A***, aggiornando gli indicatori: se si ottiene un valore diverso da zero, corrispondente al codice di conferma, si salta all'indirizzo di memoria indicato come secondo argomento.
`stop`	1	Ferma il segnale di clock per tutta la CPU.

Nelle sezioni successive vengono proposti alcuni esempi per verificare il funzionamento delle macroistruzioni della versione attuale della CPU dimostrativa. Gli esempi sono dimostrati attraverso dei video che mettono in evidenza anche l'accesso alla memoria RAM.

Macrocodice: chiamata di una routine

```
begin macrocode @ 0
start:
        load #data_3          // Colloca la pila dei dati.
        mv_mdr_sp
        call #somma           // Chiama la funzione somma().
        stop
somma:
        load #data_0
        mv_mdr_a
        load #data_1
        mv_mdr_b
        add
        mv_a_mdr
        store #data_2
        return
data_0:
        .byte 0x11
data_1:
        .byte 0x12
data_2:
        .byte 0x00
data_3:
        .byte 0x30
end
```

Figura u114.26. Situazione conclusiva del bus dati dopo l'esecuzione del codice contenuto nel listato precedente. Video: *http://www.youtube.com/watch?v=eATz3XLYWbc*

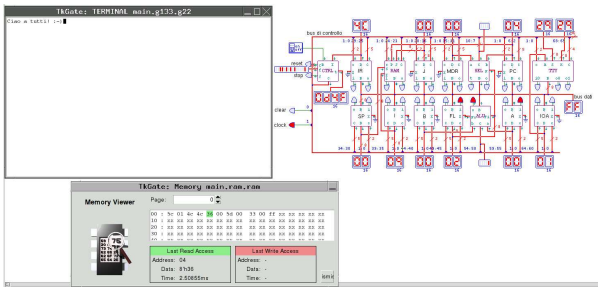


Macrocodice: inserimento da tastiera e visualizzazione sullo schermo

```
begin macrocode @ 0
start:
        in 1          // Legge dalla tastiera.
        not           // Inverte per aggiornare
        not           //   gli indicatori.
```

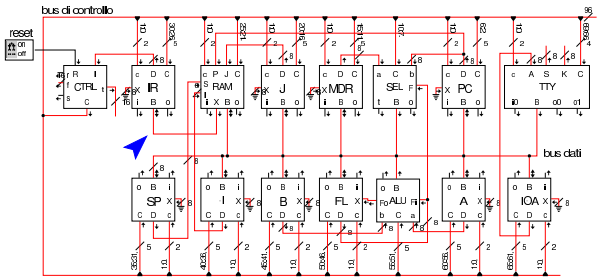
```
        jump_z #start // Se il valore è zero,
                      //   ripete la lettura.
        out 0         // Altrimenti emette lo stesso
                      //   valore sullo schermo.
        jump #start   // Ricomincia.
stop:                 // Non raggiunge mai questo punto.
        stop
end
```

Figura u114.28. Inserimento da tastiera e visualizzazione sullo schermo. Video: *http://www.youtube.com/watch?v=m22oK22ULTwWo*



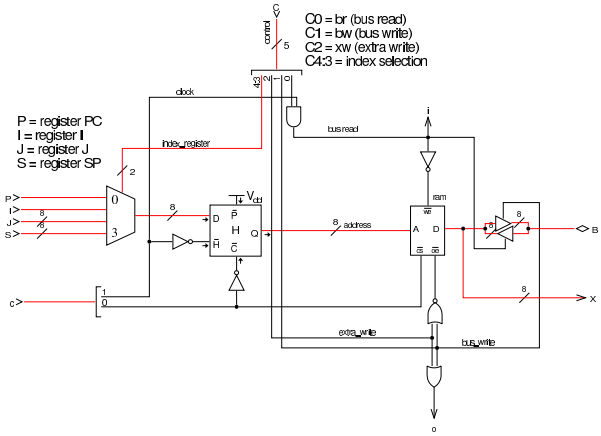
Viene proposta una modifica ulteriore della CPU dimostrativa, con lo scopo di migliorare leggermente la sua efficienza, attraverso il collegamento diretto tra il modulo **RAM** e il registro ***IR***, per non interferire con il bus dati quando si può trasferire un codice operativo dalla memoria al registro relativo.

Figura u115.1. CPU dimostrativa, versione «J».



Per realizzare il collegamento evidenziato nella figura, il modulo **RAM** viene modificato, per poter pilotare un'uscita ausiliaria (***X***); di conseguenza di ingrandisce il suo collegamento al bus di controllo, costringendo a riadattare i collegamenti degli altri moduli.

Figura u115.2. Modulo **RAM** modificato con l'aggiunta dell'uscita ausiliaria.



La dichiarazione dei campi del bus di controllo richiede quindi il cambiamento del significato delle linee di controllo relative al modulo **RAM**, assieme allo slittamento in avanti del collegamento degli altri moduli che seguono:

```
field ctrl[1:0]  = {nop=0, stop=1, load=2};
field pc[6:2]    = {br=1, bw=2, xr=4, inc=8, dec=16};
field sel[10:7]  = {if_carry=1, if_not_carry=3,
                    if_zero=5, if_not_zero=7,
                    if_negative=9, if_not_negative=11,
                    if_overflow=13, if_not_overflow=15};
field mdr[15:11] = {br=1, bw=2, xr=4, inc=8, dec=16};
field j[20:16]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field ram[25:21] = {br=1, bw=2, xw=4, p=0, i=8, j=16, s=24};
field ir[30:26]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field sp[35:31]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field i[40:36]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field b[45:41]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field fl[50:46]  = {br=1, bw=2, xr=4, inc=8, dec=16};
field alu[55:51] = {not=1, and=3, or=5, xor=7, lshl=9, lshr=11,
                    ashl=13, ashr=15, rotl=17, rotr=19, rotcl=21,
                    rotcr=23, add_c=25, sub_b=27, add=29, sub=31};
field a[60:56]   = {br=1, bw=2, xr=4, inc=8, dec=16};
field ioa[65:61] = {br=1, bw=2, xr=4, inc=8, dec=16};
field ioc[69:66] = {br=1, bw=2, req=4, isack=8};
```

Nella dichiarazione del microcodice, cambia il fatto che la RAM,

per poter comunicare con il registro ***IR***, deve avere abilitata la linea ***xr*** (*extra write*), pertanto, tutte le microistruzioni di caricamento del codice operativo successivo (*fetch*), vanno cambiate nel modo successivo:

```
    ir=br ram=xw ram=p pc=inc ctrl=load;
```

Infine, dove possibile, la microistruzione di caricamento (*fetch*) che appare alla fine di ogni macroistruzione, si fonde con la penultima macroistruzione. Si tratta precisamente delle istruzioni relative alla copia di un valore da un registro a un altro e quelle che utilizzano la ALU: si riducono tutte a un solo ciclo di clock:

```
mv_mdr_a:
    a=br mdr=bw ir=br ram=xw ram=p pc=inc ctrl=load;
mv_mdr_b:
    b=br mdr=bw ir=br ram=xw ram=p pc=inc ctrl=load;
…
mv_j_i:
    i=br j=bw ir=br ram=xw ram=p pc=inc ctrl=load;
mv_j_mdr:
    mdr=br j=bw ir=br ram=xw ram=p pc=inc ctrl=load;
```

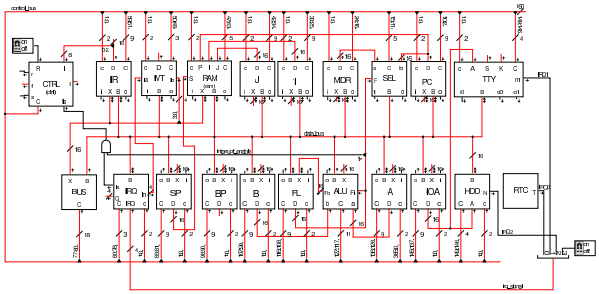
```
not:
    a=br alu=not fl=xr ir=br ram=xw ram=p pc=inc ctrl=load;
and:
    a=br alu=and fl=xr ir=br ram=xw ram=p pc=inc ctrl=load;
…
add:
    a=br alu=add fl=xr ir=br ram=xw ram=p pc=inc ctrl=load;
sub:
    a=br alu=sub fl=xr ir=br ram=xw ram=p pc=inc ctrl=load;
```


Versione K: 16 bit «little-endian»

Registri a 16 bit 911
Modulo «BUS» 913
Modulo «ALU» 913
Modulo «SEL» 916
Modulo «RAM» 917
Modulo «IRQ» 918
Modulo «IVT» 920
Modulo «CTRL» 921
Codici operativi 923
Microcodice 934
Gestione delle interruzioni 942
Orologio: modulo «RTC» 944
Modulo «TTY» 944
Modulo «HDD» 945
Macrocodice: esempio di uso del terminale con le interruzioni 947

Viene proposta un'estensione ulteriore del progetto con registri a 16 bit, pur continuando a gestire una memoria organizzata a blocchi da 8 bit. Dal momento che il compilatore di microcodice e macrocodice di Tkgate memorizza i valori a 16 bit invertendo l'ordine dei byte (o almeno lo fa nella versione compilata per architettura x86), questa versione della CPU (che ormai è un elaboratore completo di dispositivi) è organizzata in modalità *little-endian*.

Figura u116.1. CPU dimostrativa, versione «K».



Tra le varie novità, nella figura si può osservare la presenza del registro ***BP*** il cui scopo è quello di agevolare l'uso della pila dei dati quando si eseguono chiamate di funzione. Tale registro andrebbe usato in modo simile a quello con lo stesso nome e si trova nelle CPU 8086-8088.

Registri a 16 bit

I registri di questa versione della CPU dimostrativa sono da 16 bit, ma sono divisi in due byte, i cui contenuti sono accessibili separatamente. Inoltre, è possibile incrementare e ridurre il valore di tali registri, di una o di due unità.

Figura u116.2. Aspetto esterno dei registri a 16 bit.

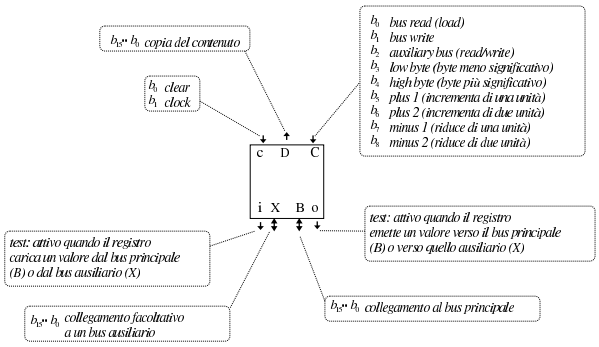


Figura u116.3. Struttura dei registri a 16 bit.

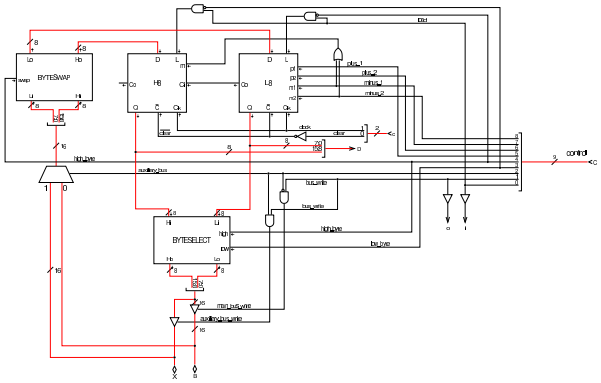
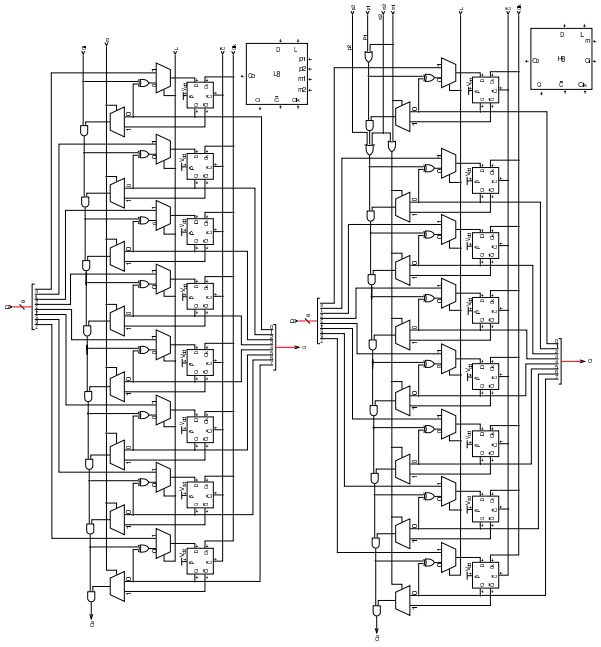


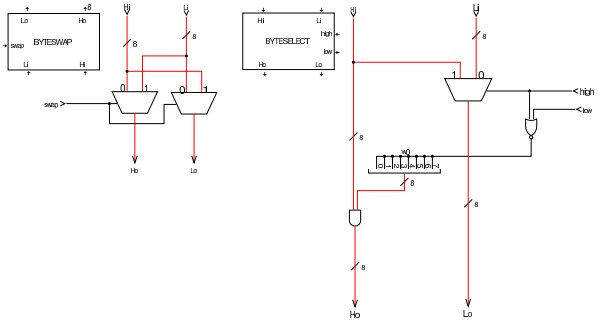
Figura u116.4. Struttura dei moduli **H8** e **L8**.



Il modulo **BYTESELECT** contenuto nei registri a 16 bit, serve a limitare la lettura del contenuto del registro a soli 8 bit, scegliendo tra la parte meno significativa o quella più significativa. Per esempio, se il registro contiene il valore $ABCD_{16}$ e si seleziona il byte meno significativo, si ottiene $00CD_{16}$, al contrario, se si seleziona il byte più significativo, si ottiene $00AB_{16}$.

Quando si inserisce un valore nel registro, è possibile scrivere solo nella la porzione inferiore o solo in quella superiore. Per questo si utilizza il modulo **BYTESWAP** che permette di scambiare i byte del valore recepito dal bus; poi sta ai moduli **L8** o **H8** attivarsi per caricare la porzione rispettiva se ciò è richiesto dai segnali del bus di controllo. In pratica, quando si sta ricevendo dal bus dati un valore a 8 bit che deve essere collocato nella porzione superiore del registro, i segnali del bus di controllo attivano lo scambio dei byte con il modulo **BYTESWAP** e attivano il caricamento dell'informazione solo nel registro ***H8***.

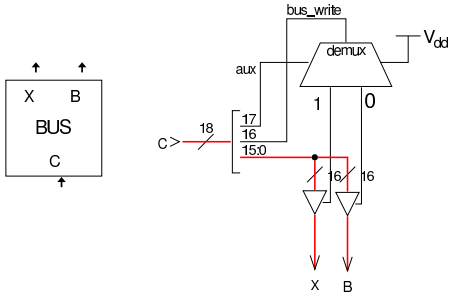
Figura u116.5. Struttura interna dei moduli **BYTESWAP** e **BYTESELECT** per lo scambio o la selezione dei byte.



Modulo «BUS»

Rispetto alla versione precedente della CPU dimostrativa, si aggiunge un modulo molto semplice che consente al sistema di controllo di inserire un valore nel bus, scegliendo tra quello principale (***B***) o quello ausiliario (***X***).

Figura u116.6. Modulo **BUS**.



Modulo «ALU»

L'unità ALU è stata ridisegnata, allo scopo di gestire valori a 16 bit e per poter disporre di qualche funzionalità in più. In particolare, si distinguono indicatori diversi per le operazioni che riguardano 8 bit rispetto a quelle che vanno intese a 16.

Figura u116.7. Struttura complessiva della ALU.

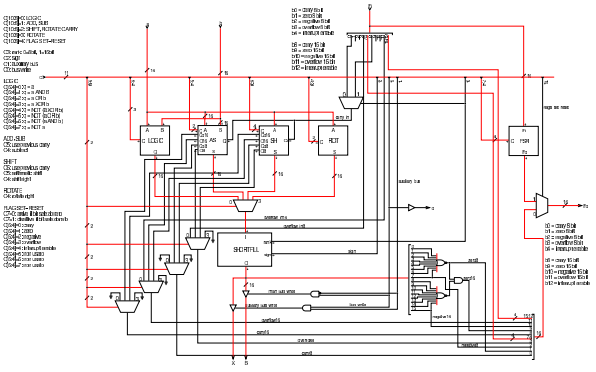


Figura u116.8. Struttura dell'unità logica interna alla ALU.

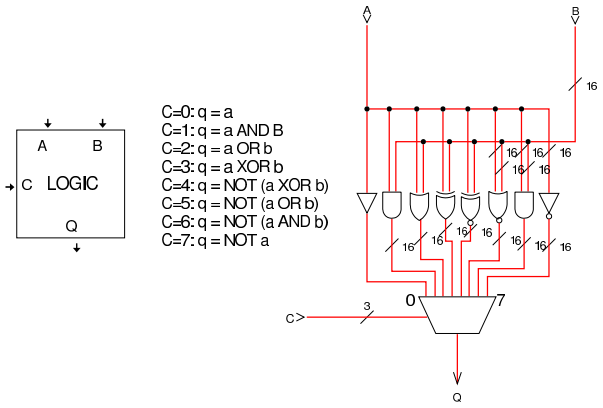


Figura u116.9. Struttura del modulo di addizione e sottrazione **AS**.

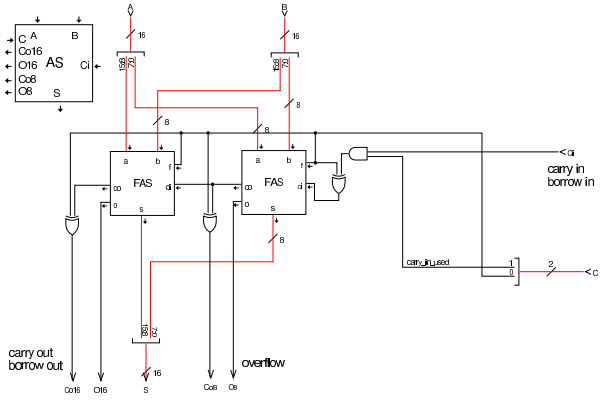


Figura u116.10. Modulo **FAS** (*full adder-subtractor*) contenuto nell'unità aritmetica. I moduli **fa** sono degli addizionatori completi (*full adder*).

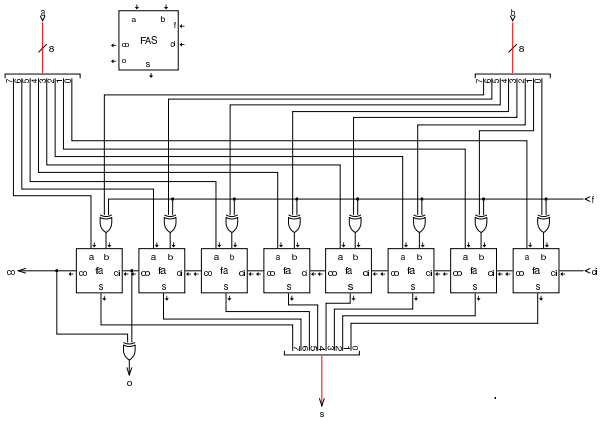


Figura u116.11. Struttura del modulo di scorrimento (**SH**), il quale si occupa anche della rotazione con riporto.

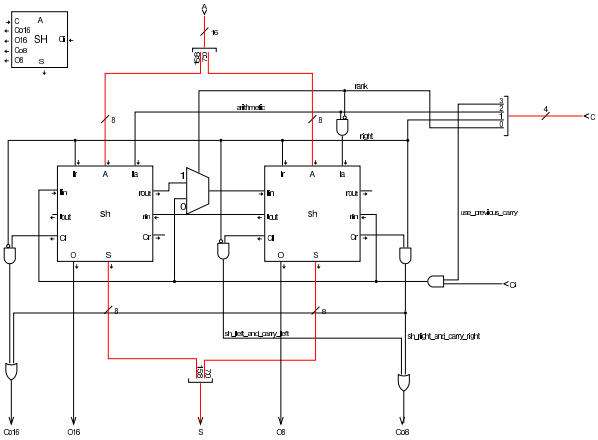


Figura u116.12. Modulo di rotazione (**ROT**): questo modulo esegue esclusivamente la rotazione del contenuto, senza utilizzare il riporto.

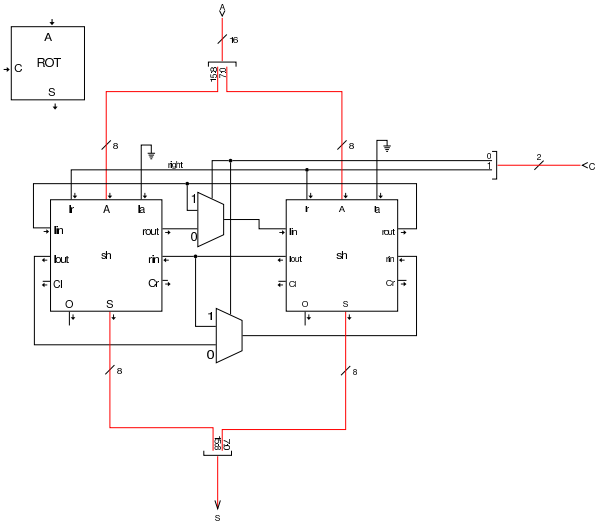
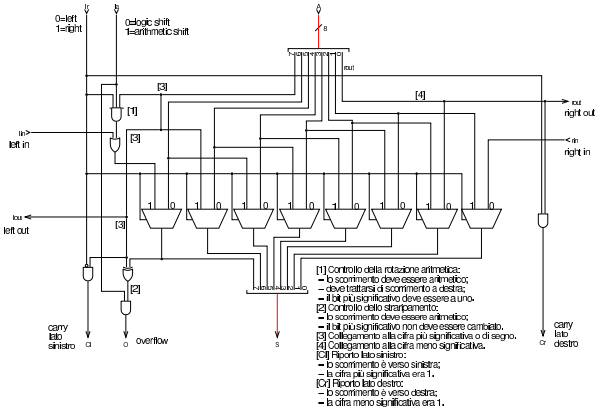


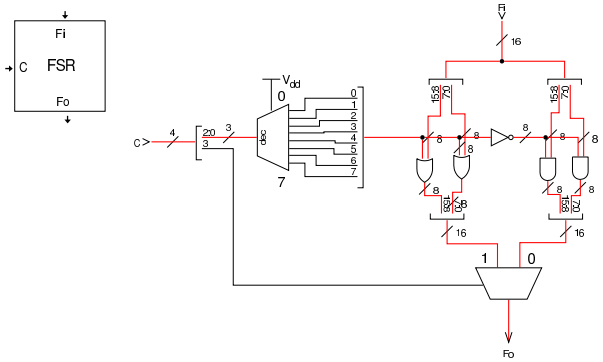
Figura u116.13. Modulo **sh** contenuto nei moduli di scorrimento e di rotazione.



La ALU di questa versione della CPU dimostrativa consente di modificare lo stato degli indicatori, attraverso il modulo **FAS** (*flags add-subtract*). Prima di tutto va osservato che gli indicatori sono doppi, su due gruppi da 8 bit, per consentire di distinguere quando alcune operazioni producono l'alterazione degli indicatori in modo diverso se si considerano valori a 8 bit o valori a 16 bit; per esempio esiste

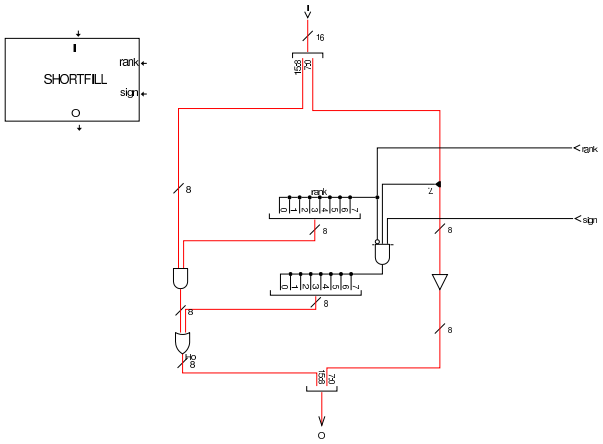
un indicatore di riporto a 8 bit e un altro a 16 bit. Quando si interviene per modificare lo stato degli indicatori, si agisce simultaneamente in entrambi i gruppi, attivandoli o disattivandoli assieme. Il modulo **FAS** riceve quindi una maschera da 8 bit e la funzione da applicare a questa maschera: si può applicare l'operatore OR o l'operatore AND e si aggiorna di conseguenza lo stato dei registri.

Figura u116.14. Modulo **FSR** per l'alterazione diretta degli indicatori.



In modo simile a quello che succede nei registri, la ALU dispone del modulo **SHORTFILL** che può essere usato per adattare un valore, quando si sa che questo va considerato a 8 bit, per estendere il segno correttamente.

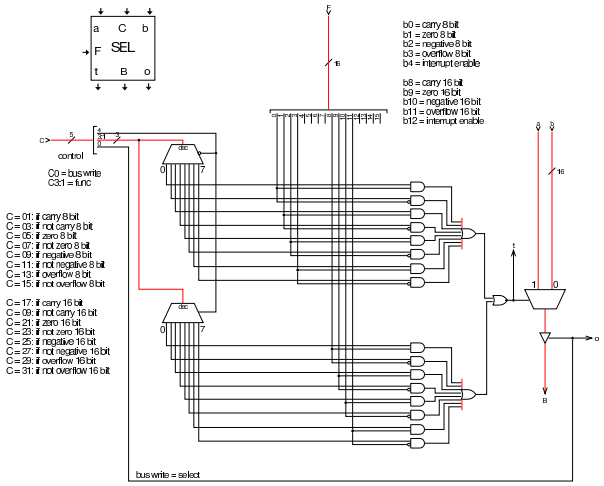
Figura u116.15. Modulo **SHORTFILL**, usato per sistemare il contenuto degli otto byte più significativi, quando si richiede di gestore solo operazioni a otto bit.



Modulo «SEL»

Il modulo **SEL** si estende per gestire gli indicatori distinti, a otto o sedici bit. Tra gli indicatori ne appare uno nuovo, relativo all'attivazione o meno delle interruzioni hardware (IRQ), ma su questo valore non si prevedono valutazioni, quindi il modulo **SEL** lo ignora.

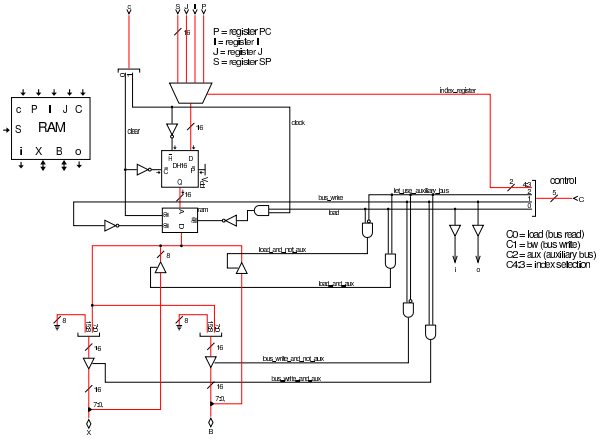
Figura u116.16. Modulo **SEL**.



Modulo «RAM»

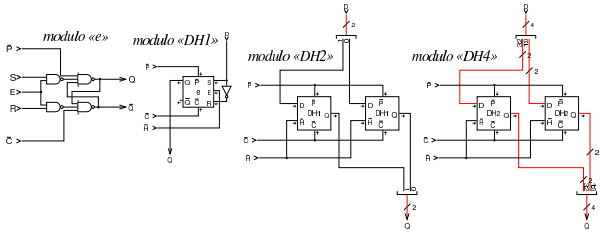
La memoria RAM continua a funzionare a blocchi di otto bit, come avviene nelle architetture comuni. Per leggere o scrivere valori a 16 bit occorre eseguire due operazioni successive; inoltre, tenendo conto che si lavora secondo l'ordine *little endian*, lettura e scrittura partono sempre dal byte meno significativo.

Figura u116.17. Modulo **RAM**.



Il modulo **RAM** contiene il registro ***DH16*** che si lascia attraversare dal valore che riceve dall'ingresso ***D*** quando l'ingresso ***H'*** è attivo, altrimenti, se ***H'*** non è attivo, mantiene in uscita il valore recepito precedentemente.

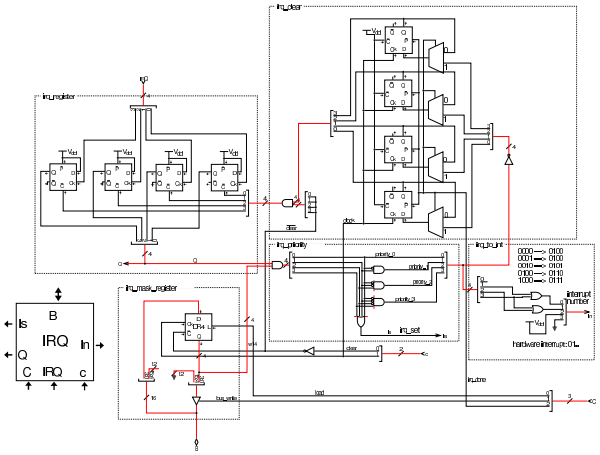
Figura u116.18. Da sinistra a destra, si vedono le fasi realizzative dei moduli **DH**...: si parte da un flip-flop SR con ingresso di abilitazione, quindi si realizza un flip-flop D con ingresso di abilitazione, poi si mettono in parallelo i flip-flop D. Si intende che il registro ***DH16*** è composto con due registri ***DH8***, il quale, a sua volta, è composto da due registri ***DH4***.



Modulo «IRQ»

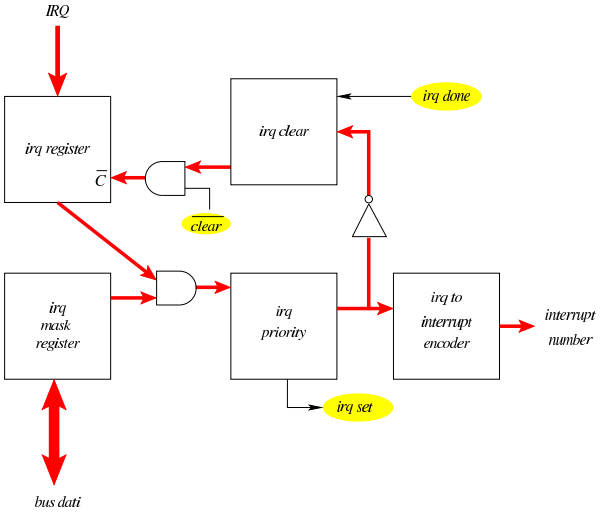
Questa versione della CPU dimostrativa gestisce le interruzioni, distinguendo tra quelle prodotte internamente dalla CPU stessa, quelle provenienti da dispositivi esterni e quelle gestite via software. Il modulo **IRQ** si occupa di ricevere le interruzioni hardware dai dispositivi per fornirle al circuito di controllo che deve poi attuare l'interruzione. In breve, il modulo **IRQ** riceve le interruzioni in modo asincrono, le memorizza e determina quale sia l'interruzione da servire per prima. Il modulo appare esternamente come se fosse un registro, in quanto deve poter ricevere una maschera delle interruzioni ammissibili; la stessa maschera può essere letta dal modulo.

Figura u116.19. Schema complessivo del modulo **IRQ**.



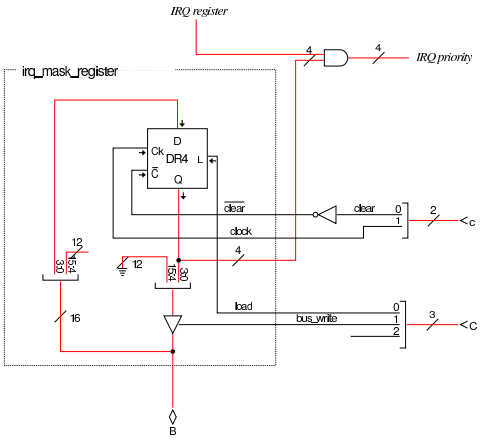
Per poter comprendere cosa fa il modulo **IRQ** è necessario analizzare i suoi vari componenti, con l'aiuto di uno schema a blocchi che riproduce in modo più semplice il suo disegno effettivo. Questo schema a blocchi è visibile nella figura successiva.

Figura u116.20. Schema a blocchi del modulo **IRQ**.



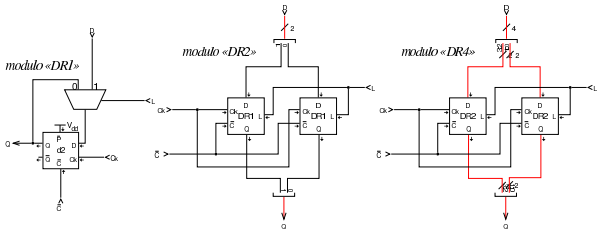
Conviene partire dall'analisi del registro che contiene la maschera degli IRQ ammissibili che appare in basso a sinistra nello schema complessivo: si tratta di un registro a 4 bit (uno per ogni IRQ gestito) che legge dal bus dati per aggiornare il proprio valore e scrive sul bus dati, per consentire di conoscere il valore che contiene (ammesso che ciò possa servire).

Figura u116.21. Dettaglio del registro della maschera degli IRQ.



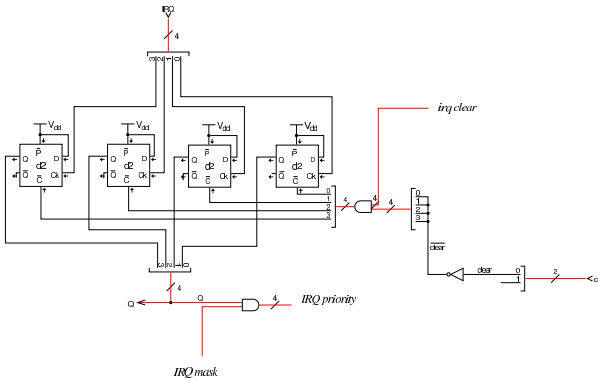
Il modulo **DR4** è un registro a quattro bit, realizzato con flip-flop D, come si vede nella figura successiva, attraverso passaggi successivi.

Figura u116.22. Costruzione dei moduli **DR**....



In alto a sinistra, nello schema generale, appare il registro degli IRQ, il cui scopo è quello di memorizzare le interruzioni hardware ricevute dall'ingresso IRQ. Questo registro è costruito in modo insolito, perché è costituito da flip-flop D a margine positivo, ma l'ingresso ***D*** di tali flip-flop è collegato in modo da essere sempre attivo, mentre l'ingresso di clock viene usato per ricevere il segnale di IRQ. In pratica, un segnale di IRQ che giunge all'ingresso clock del flip-flop, lo attiva stabilmente. I flip-flop del registro IRQ possono essere azzerati solo attraverso l'ingresso ***C'***.

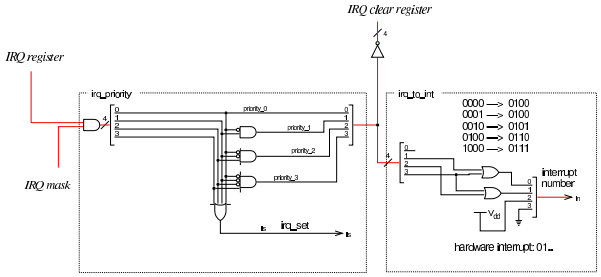
Figura u116.23. Dettaglio del registro degli IRQ ricevuti.



Il valore memorizzato nel registro IRQ e quello della maschera sottostante, vengono confrontati con una porta AND multipla (una porta distinta per ogni linea di IRQ) e quindi passati a un modulo che ne seleziona uno solo in base alla priorità: si sceglie il numero di IRQ più basso disponibile. Il modulo che ha selezionato la priorità comunica con un codificatore che si occupa di trasformare l'IRQ scelto in un numero di interruzione, per cui, IRQ0 diventa INT4, IRQ2 diventa INT5, fino a IRQ3 che diventa INT7. Si può osservare che il modulo di selezione della priorità emette un segnale (*irq set*) per informare della presenza effettiva di un IRQ che necessita di essere servito, dato che l'assenza di un IRQ produce comunque nel

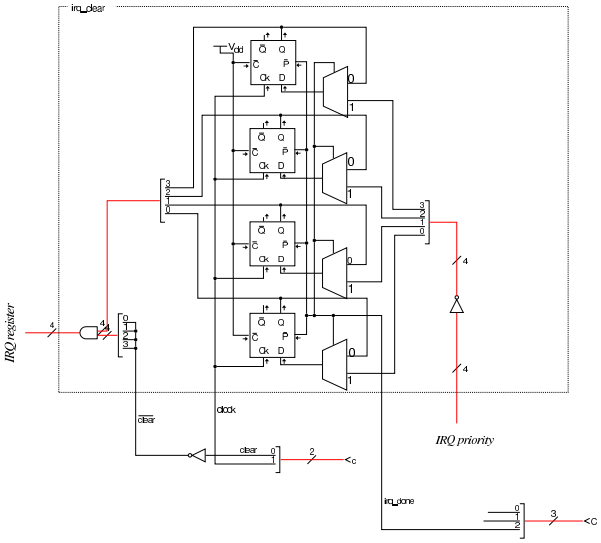
codificatore il valore INT4.

Figura u116.24. Dettaglio del selettore di priorità e del codificatore.



Quando un IRQ è stato servito, c'è la necessità di azzerare il flip-flop corrispondente nel registro degli IRQ (in alto a sinistra). Per ottenere questo risultato si utilizza il registro di azzeramento che si vede in alto a destra. Questo è composto da flip-flop D (a margine positivo) che in condizioni normali (quando l'ingresso *irq done* è pari a zero) producono in uscita un valore pari a uno, in quanto risultano inizializzati a uno (ingresso ***P'*** a zero). L'uscita di questo registro di azzeramento è collegato all'ingresso ***C'*** del registro degli IRQ, per cui, finché offre valori a uno, il registro degli IRQ mantiene il proprio valore memorizzato. Quando invece il registro di azzeramento riceve il segnale *irq done*, allora recepisce il complemento a uno del valore selezionato in base alla priorità di IRQ; in tal modo, si azzera al suo interno il bit corrispondente, azzerando di conseguenza il flip-flop del registro degli IRQ. Di conseguenza, il modulo che valuta la priorità può mettere in evidenza un altro IRQ, se disponibile.

Figura u116.25. Dettaglio del registro di cancellazione degli IRQ serviti.



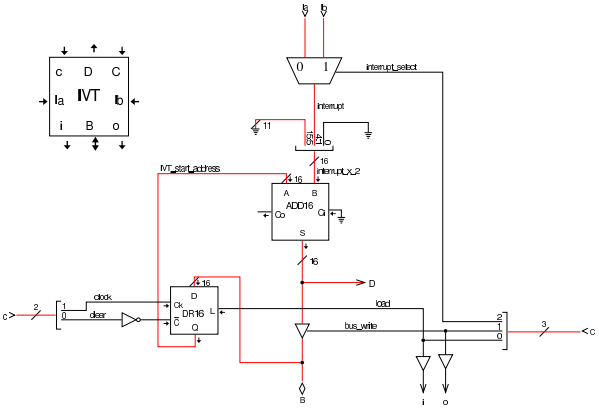
L'azzeramento del registro degli IRQ deve poter avvenire anche simultaneamente per tutti i flip-flop che contiene, pertanto il suo ingresso ***C'*** è collegato con una porta AND che consente di agire in tal modo. Il segnale ***clear'*** risulta come complemento del segnale ***clear*** proveniente dal bus di controllo.

Modulo «IVT»

Per poter gestire le interruzioni (di CPU, hardware e software), questa versione della CPU dimostrativa ha la necessità di disporre di una tabella «IVT» (*interrupt vector table*), da quale parte nella memoria RAM. La tabella IVT deve essere realizzata come un array di interi a 16 bit (*little-endian*), ognuno dei quali rappresenta l'indirizzo di una routine da eseguire quando viene attivata l'interruzione corrispondente. Pertanto, ***IVT[n]*** deve corrispondere all'indirizzo che si deve occupare di svolgere l'attività richiesta dall'interruzione ***n***.

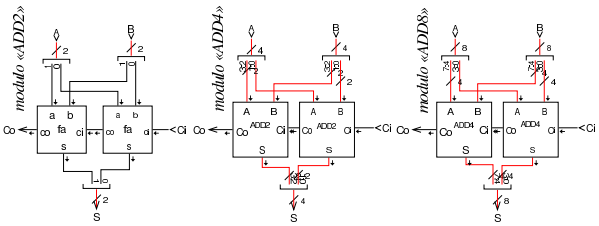
Il registro ***IVT*** serve a memorizzare la collocazione della tabella IVT, corrispondente precisamente a ***IVT[0]***. Da due ingressi indipendenti, il modulo riceve il numero di una certa interruzione, la quale viene trasformata nell'indirizzo corrispondente in memoria che contiene il riferimento alla routine da avviare.

Figura u116.26. Modulo **`IVT`**.



Nel modulo **`IVT`** si utilizza un registro **`DR16`** per memorizzare l'indirizzo di partenza della tabella IVT. Questo registro è realizzato nella stessa modalità già descritta in relazione al registro di tipo **`DR4`**, nella sezione precedente. Il modulo **`ADD16`** è composto da sedici addizionatori completi, messi in parallelo, con il riporto in cascata. Anche questo modulo viene realizzato per fasi successive, come già fatto per **`DR16`**.

Figura u116.27. Costruzione dei moduli **`ADD`**....



Modulo «CTRL»

Il modulo **`CTRL`** ha solo piccole modifiche rispetto alla versione precedente: il codice operativo rimane a otto bit (ingresso ***I***); il registro contatore (***CNT9***) è ridotto a soli nove bit, perché nel microcodice non si superano le 512 righe; le righe del microcodice richiedono molti più bit, quindi si utilizzano cinque moduli di memoria che assieme permettono di pilotare un bus di controllo da 160 bit. Nell'ingresso al contatore ***CNT9*** c'è la mediazione di un multiplatore che consente di immettere un indirizzo quando il flip-flop D che appare sulla destra è attivo. Questo indirizzo deve corrispondere al punto in cui nel microcodice si descrive la procedura necessaria a iniziare un'interruzione hardware (IRQ); in pratica deve corrispondere alla collocazione dell'etichetta '**`irq:`**', come si può determinare dai file prodotti dalla compilazione con Tkgate.

Figura u116.28. Modulo **CTRL**.

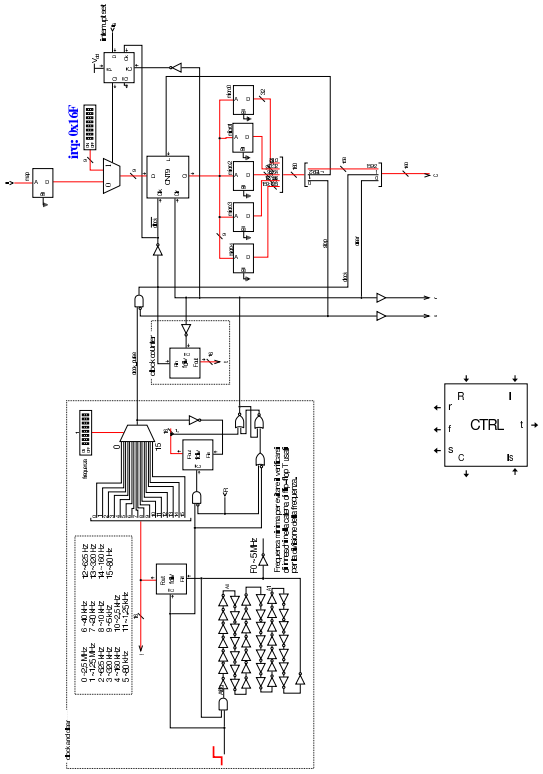
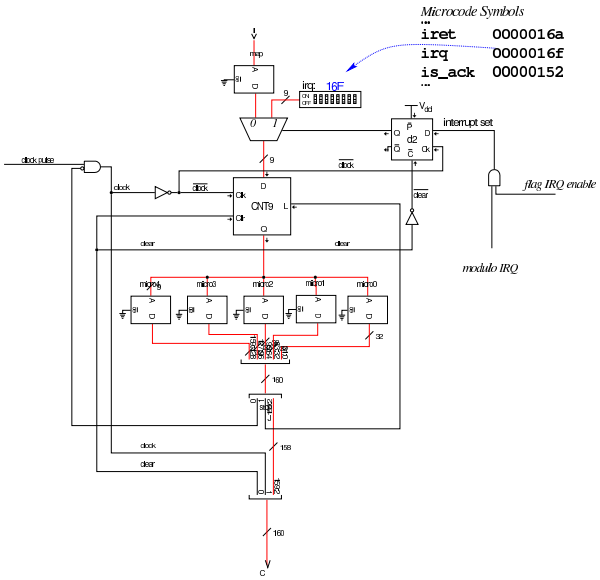


Figura u116.29. Dettaglio dell'unità di controllo che si occupa di gestire il codice.



Osservando la figura precedente, è importante chiarire cosa accade quando viene recepita un'interruzione hardware: il modulo **IRQ**, già descritto nella sezione u0.6, trasmette al modulo **IVT** il numero dell'interruzione corrispondente al numero di IRQ selezionato e attiva la propria uscita ***Is*** (*IRQ set*) che arriva all'unità di controllo solo se è attivo anche l'indicatore ***I*** (*interrupt enable*). Se le cose stanno proprio così, questa richiesta viene memorizzata nel flip-flop D (a margine positivo) che appare in alto a destra nello schema; quindi, alla prima occasione in cui l'unità di controllo deve eseguire un nuovo codice operativo, si trova invece diretta a eseguire le istruzioni corrispondenti all'etichetta '**irq:**'.

Listato u116.30. Dichiarazione delle memorie utilizzate.

```
map       bank[7:0]     ctrl.map;
microcode bank[31:0]    ctrl.micro0;
microcode bank[63:32]   ctrl.micro1;
microcode bank[95:64]   ctrl.micro2;
microcode bank[127:96]  ctrl.micro3;
microcode bank[159:128] ctrl.micro4;
macrocode bank[15:0]    ram.ram;
```

Codici operativi

I codici operativi usati in questa versione della CPU dimostrativa, utilizzano sempre solo otto bit, ma invece di usare semplicemente un numero sequenziale per distinguerli, si va a strutturare con un certo criterio lo spazio binario disponibile. Per prima cosa si definisce una conversione tra i registri utilizzabili nella programmazione e un numero intero, in modo tale da usare tre bit per la loro distinzione:

```
registers I=0, J=1, A=2, B=3, BP=4, SP=5, MDR=6, FL=7;
```

Gli operandi associati ai codici operativi sono di vario tipo; i casi più semplici sono dichiarati all'inizio:

```
operands op_0 {
    -  = { };
};
operands op_8 {
    #1 = { +1=#1[7:0]; };
};
operands op_16 {
    #1 = { +1=#1[7:0]; +2=#1[15:8]; };
};
```

Si comprende, intuitivamente, che '**op_0**' rappresenti la mancanza di operandi, che '**op_8**' rappresenti un operando di soli 8 bit, e che '**op_16**' rappresenti un operando da 16 bit, tenendo conto che la memoria RAM è però organizzata in blocchi da otto bit e l'accesso alla stessa avviene in modalità *little endian* (quindi il byte meno significativo si trova prima di quello più significativo).

Listato u116.33. Dichiarazione dei codici operativi.

```
op nop {
  map nop:       0x00;  // not operate
  +0[7:0]=0x00;
  operands op_0;
};
op mv {                 // 00...... = mv
  map nop:       0x00;  // 00000000 = mv %I %I non valido => nop
  map mv_i_j:    0x01;  // 00000001 = mv %I %J
  map mv_i_a:    0x02;  // 00000010 = mv %I %A
  map mv_i_b:    0x03;  // 00000011 = mv %I %B
  map mv_i_bp:   0x04;  // 00000100 = mv %I %BP
  map mv_i_sp:   0x05;  // 00000101 = mv %I %SP
  map mv_i_mdr:  0x06;  // 00000110 = mv %I %MDR
  map mv_i_fl:   0x07;  // 00000111 = mv %I %FL
  map mv_j_i:    0x08;  // 00001000 = mv %J %I
  map op_error:  0x09;  // 00001001 = mv %J %J non valido
  map mv_j_a:    0x0A;  // 00001010 = mv %J %A
  map mv_j_b:    0x0B;  // 00001011 = mv %J %B
  map mv_j_bp:   0x0C;  // 00001100 = mv %J %BP
  map mv_j_sp:   0x0D;  // 00001101 = mv %J %SP
  map mv_j_mdr:  0x0E;  // 00001110 = mv %J %MDR
  map mv_j_fl:   0x0F;  // 00001111 = mv %J %FL
  map mv_a_i:    0x10;  // 00010000 = mv %A %I
  map mv_a_j:    0x11;  // 00010001 = mv %A %J
  map op_error:  0x12;  // 00010010 = mv %A %A non valido
  map mv_a_b:    0x13;  // 00010011 = mv %A %B
  map mv_a_bp:   0x14;  // 00010100 = mv %A %BP
  map mv_a_sp:   0x15;  // 00010101 = mv %A %SP
  map mv_a_mdr:  0x16;  // 00010110 = mv %A %MDR
  map mv_a_fl:   0x17;  // 00010111 = mv %A %FL
  map mv_b_i:    0x18;  // 00011000 = mv %B %I
  map mv_b_j:    0x19;  // 00011001 = mv %B %J
  map mv_b_a:    0x1A;  // 00011010 = mv %B %A
  map op_error:  0x1B;  // 00011011 = mv %B %B non valido
  map mv_b_bp:   0x1C;  // 00011100 = mv %B %BP
  map mv_b_sp:   0x1D;  // 00011101 = mv %B %SP
  map mv_b_mdr:  0x1E;  // 00011110 = mv %B %MDR
  map mv_b_fl:   0x1F;  // 00011111 = mv %B %FL
  map mv_bp_i:   0x20;  // 00100000 = mv %BP %I
  map mv_bp_j:   0x21;  // 00100001 = mv %BP %J
```

```
  map mv_bp_a:   0x22;  // 00100010 = mv %BP %A
  map mv_bp_b:   0x23;  // 00100011 = mv %BP %B
  map op_error:  0x24;  // 00100100 = mv %BP %BP non valido
  map mv_bp_sp:  0x25;  // 00100101 = mv %BP %SP
  map mv_bp_mdr: 0x26;  // 00100110 = mv %BP %MDR
  map mv_bp_fl:  0x27;  // 00100111 = mv %BP %FL
  map mv_sp_i:   0x28;  // 00101000 = mv %SP %I
  map mv_sp_j:   0x29;  // 00101001 = mv %SP %J
  map mv_sp_a:   0x2A;  // 00101010 = mv %SP %A
  map mv_sp_b:   0x2B;  // 00101011 = mv %SP %B
  map mv_sp_bp:  0x2C;  // 00101100 = mv %SP %BP
  map op_error:  0x2D;  // 00101101 = mv %SP %SP non valido
  map mv_sp_mdr: 0x2E;  // 00101110 = mv %SP %MDR
  map mv_sp_fl:  0x2F;  // 00101111 = mv %SP %FL
  map mv_mdr_i:  0x30;  // 00110000 = mv %MDR %I
  map mv_mdr_j:  0x31;  // 00110001 = mv %MDR %J
  map mv_mdr_a:  0x32;  // 00110010 = mv %MDR %A
  map mv_mdr_b:  0x33;  // 00110011 = mv %MDR %B
  map mv_mdr_bp: 0x34;  // 00110100 = mv %MDR %BP
  map mv_mdr_sp: 0x35;  // 00110101 = mv %MDR %SP
  map op_error:  0x36;  // 00110110 = mv %MDR %MDR non valido
  map mv_mdr_fl: 0x37;  // 00110111 = mv %MDR %FL
  map mv_fl_i:   0x38;  // 00111000 = mv %FL %I
  map mv_fl_j:   0x39;  // 00111001 = mv %FL %J
  map mv_fl_a:   0x3A;  // 00111010 = mv %FL %A
  map mv_fl_b:   0x3B;  // 00111011 = mv %FL %B
  map mv_fl_bp:  0x3C;  // 00111100 = mv %FL %BP
  map mv_fl_sp:  0x3D;  // 00111101 = mv %FL %SP
  map mv_fl_mdr: 0x3E;  // 00111110 = mv %FL %MDR
  map op_error:  0x3F;  // 00111111 = mv %FL %FL non valido
  +0[7:0]=0x00;
  operands {
    %1,%2 = { +0[5:3]=%1; +0[2:0]=%2; };
  };
};
op load8 {              // 010001.. = load8
  map load8_i:   0x44;  // 01000100 = load8 %I
  map load8_j:   0x45;  // 01000101 = load8 %J
  map load8:     0x46;  // 01000110 = load8 #...
  +0[7:0]=0x44;
  operands {
    %1 = { +0[1]=0; +0[0]=%1; };
    #1 = { +0[1]=1; +0[0]=0; +1=#1[7:0]; +2=#1[15:8]; };
  };
};
op load16 {             // 010010.. = load16
  map load16_i:  0x48;  // 01001000 = load16 %I
  map load16_j:  0x49;  // 01001001 = load16 %J
  map load16:    0x4A;  // 01001010 = load16 #...
  +0[7:0]=0x48;
  operands {
    %1 = { +0[1]=0; +0[0]=%1; };
    #1 = { +0[1]=1; +0[0]=0; +1=#1[7:0]; +2=#1[15:8]; };
  };
};
op store8 {             // 010011.. = store8
  map store8_i:  0x4C;  // 01001100 = store8 %I
  map store8_j:  0x4D;  // 01001101 = store8 %J
  map store8:    0x4E;  // 01001110 = store8 #...
  +0[7:0]=0x4C;
  operands {
    %1 = { +0[1]=0; +0[0]=%1; };
    #1 = { +0[1]=1; +0[0]=0; +1=#1[7:0]; +2=#1[15:8]; };
  };
};
op store16 {            // 010100.. = store16
  map store16_i: 0x50;  // 01010000 = store16 %I
  map store16_j: 0x51;  // 01010001 = store16 %J
  map store16:   0x52;  // 01010010 = store16 #...
  +0[7:0]=0x50;
  operands {
    %1 = { +0[1]=0; +0[0]=%1; };
    #1 = { +0[1]=1; +0[0]=0; +1=#1[7:0]; +2=#1[15:8]; };
  };
};
op cp8 {                // 0101010. = cp8
  map cp8_ij:    0x54;  // 01010100 = cp8 %I
  map cp8_ji:    0x55;  // 01010101 = cp8 %J
  +0[7:0]=0x54;
  operands {
    %1 = { +0[0]=%1; };
  };
};
op cp16 {               // 0101011. = cp16
  map cp16_ij:   0x56;  // 01010110 = cp16 %I
  map cp16_ji:   0x57;  // 01010111 = cp16 %J
```



```
  +0[7:0]=0x56;
  operands {
    %1 = { +0[0]=%1; };
  };
};
op return {             // 010110..
  map return:    0x58;  // 01011000 = return
  +0[7:0]=0x58;
  operands op_0;
};
op call {               // 010110..
  map call:      0x59;  // 01011001 = call #...
  map call_i:    0x5A;  // 01011010 = call %I
  map call_j:    0x5B;  // 01011011 = call %J
  +0[7:0]=0x58;
  operands {
    #1 = { +0[1]=0; +0[0]=1; +1=#1[7:0]; +2=#1[15:8]; };
    %1 = { +0[1]=1; +0[0]=%1; };
  };
};
op int {
  map int:       0x5C;  // 01011100 = int #...
  +0[7:0]=0x5C;
  operands op_8;
};
op iret {
  map iret:      0x5D;  // 01011101 = iret
  +0[7:0]=0x5D;
  operands op_0;
};
op cleari {
  map cleari:    0x5E;  // 01011110 = clear interrupt flag
  +0[7:0]=0x5E;
  operands op_0;
};
op seti {
  map seti:      0x5F;  // 01011111 = set interrupt flag
  +0[7:0]=0x5F;
  operands op_0;
};
op ivtl {
  map ivtl:      0x60;  // 01100000 = load IVT location
  +0[7:0]=0x60;
  operands op_16;
};
op jump {
  map jump:      0x61;  // 01100001 = jump #...
  +0[7:0]=0x61;
  operands op_16;
};
op jump8c {
  map jump8c:    0x62;  // 01100010 = jump8c #...
  +0[7:0]=0x62;
  operands op_16;
};
op jump8nc {
  map jump8nc:   0x63;  // 01100011 = jump8nc #...
  +0[7:0]=0x63;
  operands op_16;
};
op jump8z {
  map jump8z:    0x64;  // 01100100 = jump8z #...
  +0[7:0]=0x64;
  operands op_16;
};
op jump8nz {
  map jump8nz:   0x65;  // 01100101 = jump8nz #...
  +0[7:0]=0x65;
  operands op_16;
};
op jump8o {
  map jump8o:    0x66;  // 01100110 = jump8o #...
  +0[7:0]=0x66;
  operands op_16;
};
op jump8no {
  map jump8no:   0x67;  // 01100111 = jump8no #...
  +0[7:0]=0x67;
  operands op_16;
};
op jump8n {
  map jump8n:    0x68;  // 01101000 = jump8n #...
  +0[7:0]=0x66;
  operands op_16;
};
op jump8nn {
```

```
  map jump8nn:   0x69;  // 01101001 = jump8nn #...
  map op_error:  0x6A;  // 01101010 = non valido
  map op_error:  0x6B;  // 01101011 = non valido
  map op_error:  0x6C;  // 01101100 = non valido
  map op_error:  0x6D;  // 01101101 = non valido
  map op_error:  0x6E;  // 01101110 = non valido
  map op_error:  0x6F;  // 01101111 = non valido
  map op_error:  0x70;  // 01110000 = non valido
  map op_error:  0x71;  // 01110001 = non valido
  +0[7:0]=0x67;
  operands op_16;
};
op jump16c {
  map jump16c:   0x72;  // 01110010 = jump16c #...
  +0[7:0]=0x72;
  operands op_16;
};
op jump16nc {
  map jump16nc:  0x73;  // 01110011 = jump16nc #...
  +0[7:0]=0x73;
  operands op_16;
};
op jump16z {
  map jump16z:   0x74;  // 01110100 = jump16z #...
  +0[7:0]=0x74;
  operands op_16;
};
op jump16nz {
  map jump16nz:  0x75;  // 01110101 = jump16nz #...
  +0[7:0]=0x75;
  operands op_16;
};
op jump16o {
  map jump16o:   0x76;  // 01110110 = jump16o #...
  +0[7:0]=0x76;
  operands op_16;
};
op jump16no {
  map jump16no:  0x77;  // 01110111 = jump16no #...
  +0[7:0]=0x77;
  operands op_16;
};
op jump16n {
  map jump16n:   0x78;  // 01111000 = jump16n #...
  +0[7:0]=0x76;
  operands op_16;
};
op jump16nn {
  map jump16nn:  0x79;  // 01111001 = jump16 #...
  map op_error:  0x7A;  // 01111010 = non valido
  map op_error:  0x7B;  // 01111011 = non valido
  map op_error:  0x7C;  // 01111100 = non valido
  map op_error:  0x7D;  // 01111101 = non valido
  map op_error:  0x7E;  // 01111110 = non valido
  map op_error:  0x7F;  // 01111111 = non valido
  +0[7:0]=0x77;
  operands op_16;
};
op push8 {              // 10000... = push8
  map push8_i:   0x80;  // 10000000 = push8 %I
  map push8_j:   0x81;  // 10000001 = push8 %J
  map push8_a:   0x82;  // 10000010 = push8 %A
  map push8_b:   0x83;  // 10000011 = push8 %B
  map push8_bp:  0x84;  // 10000100 = push8 %BP
  map op_error:  0x85;  // 10000101 = push8 %SP non valido
  map push8_mdr: 0x86;  // 10000110 = push8 %MDR
  map push8_fl:  0x87;  // 10000111 = push8 %FL
  +0[7:0]=0x80;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
op pop8 {               // 10001... = pop8
  map pop8_i:    0x88;  // 10001000 = pop8 %I
  map pop8_j:    0x89;  // 10001001 = pop8 %J
  map pop8_a:    0x8A;  // 10001010 = pop8 %A
  map pop8_b:    0x8B;  // 10001011 = pop8 %B
  map pop8_bp:   0x8C;  // 10001100 = pop8 %BP
  map op_error:  0x8D;  // 10001101 = pop8 %SP non valido
  map pop8_mdr:  0x8E;  // 10001110 = pop8 %MDR
  map pop8_fl:   0x8F;  // 10001111 = pop8 %FL
  +0[7:0]=0x88;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
```



```
op push16 {             // 10010... = push16
  map push16_i:   0x90; // 10010000 = push16 %I
  map push16_j:   0x91; // 10010001 = push16 %J
  map push16_a:   0x92; // 10010010 = push16 %A
  map push16_b:   0x93; // 10010011 = push16 %B
  map push16_bp:  0x94; // 10010100 = push16 %BP
  map op_error:   0x95; // 10010101 = push16 %SP non valido
  map push16_mdr: 0x96; // 10010110 = push16 %MDR
  map push16_fl:  0x97; // 10010111 = push16 %FL
  +0[7:0]=0x90;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
op pop16 {              // 10011... = pop16
  map pop16_i:   0x98;  // 10011000 = pop16 %I
  map pop16_j:   0x99;  // 10011001 = pop16 %J
  map pop16_a:   0x9A;  // 10011010 = pop16 %A
  map pop16_b:   0x9B;  // 10011011 = pop16 %B
  map pop16_bp:  0x9C;  // 10011100 = pop16 %BP
  map op_error:  0x9D;  // 10011101 = pop16 %SP non valido
  map pop16_mdr: 0x9E;  // 10011110 = pop16 %MDR
  map pop16_fl:  0x9F;  // 10011111 = pop16 %FL
  +0[7:0]=0x98;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
op c8to16u {
  map c8to16u:   0xA0;  // 10100000
  +0[7:0]=0xA0;
  operands op_0;
};
op c8to16s {
  map c8to16s:   0xA1;  // 10100001
  +0[7:0]=0xA1;
  operands op_0;
};
op equal {
  map equal:     0xA2;  // 10100010
  +0[7:0]=0xA2;
  operands op_0;
};
op not {
  map not:       0xA3;  // 10100011
  +0[7:0]=0xA3;
  operands op_0;
};
op and {
  map and:       0xA4;  // 10100100
  +0[7:0]=0xA4;
  operands op_0;
};
op nand {
  map nand:      0xA5;  // 10100101
  +0[7:0]=0xA5;
  operands op_0;
};
op or {
  map or:        0xA6;  // 10100110
  +0[7:0]=0xA6;
  operands op_0;
};
op nor {
  map nor:       0xA7;  // 10100111
  +0[7:0]=0xA7;
  operands op_0;
};
op xor {
  map xor:       0xA8;  // 10101000
  +0[7:0]=0xA8;
  operands op_0;
};
op nxor {
  map nxor:      0xA9;  // 10101001
  +0[7:0]=0xA9;
  operands op_0;
};
op add {
  map add:       0xAA;  // 10101010
  +0[7:0]=0xAA;
  operands op_0;
};
op sub {
  map sub:       0xAB;  // 10101011
  +0[7:0]=0xAB;
```

```
  operands op_0;
};
op addc8 {
  map addc8:     0xAC;  // 10101100
  +0[7:0]=0xAC;
  operands op_0;
};
op subb8 {
  map subb8:     0xAD;  // 10101101
  +0[7:0]=0xAD;
  operands op_0;
};
op addc16 {
  map addc16:    0xAE;  // 10101110
  +0[7:0]=0xAE;
  operands op_0;
};
op subb16 {
  map subb16:    0xAF;  // 10101111
  +0[7:0]=0xAF;
  operands op_0;
};
op lshl8 {
  map lshl8:     0xB0;  // 10110000
  +0[7:0]=0xB0;
  operands op_0;
};
op lshr8 {
  map lshr8:     0xB1;  // 10110001
  +0[7:0]=0xB1;
  operands op_0;
};
op ashl8 {
  map ashl8:     0xB2;  // 10110010
  +0[7:0]=0xB2;
  operands op_0;
};
op ashr8 {
  map ashr8:     0xB3;  // 10110011
  +0[7:0]=0xB3;
  operands op_0;
};
op rotcl8 {
  map rotcl8:    0xB4;  // 10110100
  +0[7:0]=0xB4;
  operands op_0;
};
op rotcr8 {
  map rotcr8:    0xB5;  // 10110101
  +0[7:0]=0xB5;
  operands op_0;
};
op rotl8 {
  map rotl8:     0xB6;  // 10110110
  +0[7:0]=0xB6;
  operands op_0;
};
op rotr8 {
  map rotr8:     0xB7;  // 10110111
  +0[7:0]=0xB7;
  operands op_0;
};
op lshl16 {
  map lshl16:    0xB8;  // 10111000
  +0[7:0]=0xB8;
  operands op_0;
};
op lshr16 {
  map lshr16:    0xB9;  // 10111001
  +0[7:0]=0xB9;
  operands op_0;
};
op ashl16 {
  map ashl16:    0xBA;  // 10111010
  +0[7:0]=0xBA;
  operands op_0;
};
op ashr16 {
  map ashr16:    0xBB;  // 10111011
  +0[7:0]=0xBB;
  operands op_0;
};
op rotcl16 {
  map rotcl16:   0xBC;  // 10111100
  +0[7:0]=0xBC;
  operands op_0;
```



```
};
op rotcr16 {
  map rotcr16:   0xBD;  // 10111101
  +0[7:0]=0xBD;
  operands op_0;
};
op rotl16 {
  map rotl16:    0xBE;  // 10111110
  +0[7:0]=0xBE;
  operands op_0;
};
op rotr16 {
  map rotr16:    0xBF;  // 10111111
  +0[7:0]=0xBF;
  operands op_0;
};

op in {
  map in:        0xC0;  // 11000000
  +0[7:0]=0xC0;
  operands op_8;
};
op out {
  map out:       0xC1;  // 11000001
  +0[7:0]=0xC1;
  operands op_8;
};
op is_ack {
  map is_ack:    0xC2;  // 11000010
  map op_error:  0xC3;  // 11000011
  +0[7:0]=0xC2;
  operands {
    #1,#2 = { +1=#1[7:0]; +2=#1[7:0]; +3=#2[17:8]; };
  };
};
op clearc {
  map clearc:    0xC4;  // 11000100
  +0[7:0]=0xC4;
  operands op_0;
};
op setc {
  map setc:      0xC5;  // 11000101
  +0[7:0]=0xC5;
  operands op_0;
};
op cmp {
  map cmp:       0xC6;  // 11000110
  +0[7:0]=0xC6;
  operands op_0;
};
op test {
  map test:      0xC7;  // 11000111
  +0[7:0]=0xC7;
  operands op_0;
};
op imrl {
  map imrl:      0xC8;  // 11001000     // IMR load
  map op_error:  0xC9;  // 11001001
  map op_error:  0xCA;  // 11001010
  map op_error:  0xCB;  // 11001011
  map op_error:  0xCC;  // 11001100
  map op_error:  0xCD;  // 11001101
  map op_error:  0xCE;  // 11001110
  map op_error:  0xCF;  // 11001111
  +0[7:0]=0xC8;
  operands op_8;
};
op inc {
  map inc_i:     0xD0;  // 11010000     // inc %I
  map inc_j:     0xD1;  // 11010001     // inc %J
  map inc_a:     0xD2;  // 11010010     // inc %A
  map inc_b:     0xD3;  // 11010011     // inc %B
  map inc_bp:    0xD4;  // 11010100     // inc %BP
  map inc_sp:    0xD5;  // 11010101     // inc %SP
  map inc_mdr:   0xD6;  // 11010110     // inc %MDR
  map inc_fl:    0xD7;  // 11010111     // inc %FL
  +0[7:0]=0xD0;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
op dec {
  map dec_i:     0xD8;  // 11011000     // dec %I
  map dec_j:     0xD9;  // 11011001     // dec %J
  map dec_a:     0xDA;  // 11011010     // dec %A
  map dec_b:     0xDB;  // 11011011     // dec %B
```

```
  map dec_bp:    0xDC;  // 11011100     // dec %BP
  map dec_sp:    0xDD;  // 11011101     // dec %SP
  map dec_mdr:   0xDE;  // 11011110     // dec %MDR
  map dec_fl:    0xDF;  // 11011111     // dec %FL
  map op_error:  0xE0;  // 11100001
  map op_error:  0xE1;  // 11100010
  map op_error:  0xE2;  // 11100011
  map op_error:  0xE4;  // 11100100
  map op_error:  0xE5;  // 11100101
  map op_error:  0xE6;  // 11100110
  map op_error:  0xE7;  // 11100111
  map op_error:  0xE8;  // 11101000
  map op_error:  0xE9;  // 11101001
  map op_error:  0xEA;  // 11101010
  map op_error:  0xEB;  // 11101011
  map op_error:  0xEC;  // 11101100
  map op_error:  0xED;  // 11101101
  map op_error:  0xEE;  // 11101110
  map op_error:  0xEF;  // 11101111
  map op_error:  0xF0;  // 11110001
  map op_error:  0xF1;  // 11110010
  map op_error:  0xF2;  // 11110011
  map op_error:  0xF4;  // 11110100
  map op_error:  0xF5;  // 11110101
  map op_error:  0xF6;  // 11110110
  map op_error:  0xF7;  // 11110111
  map op_error:  0xF8;  // 11111000
  map op_error:  0xF9;  // 11111001
  map op_error:  0xFA;  // 11111010
  map op_error:  0xFB;  // 11111011
  map op_error:  0xFC;  // 11111100
  map op_error:  0xFD;  // 11111101
  map op_error:  0xFE;  // 11111110
  +0[7:0]=0xD8;
  operands {
    %1 = { +0[2:0]=%1; };
  };
};
```

Tabella u116.34. Elenco completo dei codici operativi con le macroistruzioni corrispondenti.

Sintassi del macrocodice	Codice operativo binario	Descrizione
`nop`	00000000	Non fa alcunché.
`mv %`***src***`, %`***dst***	`00`***sss******ddd***	Copia il contenuto del registro ***src*** nel registro ***dst***; nel codice operativo i bit ***sss*** rappresentano il primo registro, mentre i bit ***ddd*** rappresentano il secondo. Tra i codici operativi sono esclusi quelli che copierebbero lo stesso registro su se stesso; pertanto, il codice 00000000_2 rimane destinato all'istruzione '**nop**', in quanto rappresenterebbe la copia del registro ***A*** su se stesso.

`load8 %`***indice***
`load8 #`***indice***
`load16 %`***indice***
`load16 #`***indice***
	`0100010`***i***
	`01000110`
	`0100100`***i***
	`01001010`
		Carica nel registro ***MDR*** un valore a 8 o 16 bit dalla memoria. L'argomento può essere un registro indice (***I*** o ***J***) e in tal caso si utilizza il bit ***i*** del codice operativo per distinguerlo; altrimenti può essere direttamente l'indirizzo della memoria a cui ci si riferisce e i due bit meno significativi del codice operativo sono pari a 10_2.

Sintassi del macrocodice	Codice operativo binario	Descrizione

`store8 %`***indice***
`store8 #`***indice***
`store16 %`***indice***
`store16 #`***indice***
	`0100110`***i***
	`01001110`
	`0101000`***i***
	`01010010`
		Registra in memoria (8 o 16 bit), all'indirizzo corrispondente all'argomento, il valore contenuto nel registro ***MDR***. Quando l'argomento è un registro, può trattarsi solo di ***I*** o ***J*** e tale informazione si colloca nel bit ***i*** del codice operativo. Se l'argomento è rappresentato invece un numero, la parte finale del codice operativo diventa 10_2.

`cp8 %`***src*** `cp16 %`***src***	`0101010`***s*** `0101011`***s***	Copia in memoria, dalla posizione indicata nel registro indice ***src*** (che può essere ***I*** o ***J***), alla posizione indicata dall'altro registro indice, 8 o 16 bit, incrementando successivamente i due registri indice.

`return`
`call #`***indirizzo***
`call #`***indice***
	`01011000`
	`01011001`
	`0101101`***i***
		Uscita e chiamata di una routine. Il bit ***i*** rappresenta un registro indice (***I*** o ***J***).

`int #`***n_interrupt*** `iret`	`01011100` `01011101`	Esegue una chiamata di interruzione e il ritorno dalla stessa; il numero ***n_interrupt*** è a 8 bit, ma può andare solo da 0 a 16. Nella chiamata vengono salvati nella pila dei dati il registro ***FL*** e il registro ***PC***, quindi nel registro ***FL*** vengono disattivati i bit che consentono la generazione di interruzioni hardware (IRQ); al ritorno dalla chiamata, vengono ripristinati il registro ***PC*** e il registro ***FL***.
`cleari` `seti`	`01011110` `01011111`	Azzera o attiva i bit di abilitazione delle interruzioni hardware (IRQ) contenuti nel registro ***FL***.
`ivtl #`***indirizzo***	`01100000`	Carica nel registro ***IVT*** l'indirizzo della tabella IVT (*interrupt vector table*) in memoria.
`jump #`***indirizzo***	`01100001`	Salta all'esecuzione dell'istruzione contenuta nell'indirizzo indicato. L'argomento è a 16 bit.

Sintassi del macrocodice	Codice operativo binario	Descrizione
`jump8c #`***indirizzo***	`01100010`	Salta all'esecuzione dell'istruzione contenuta nell'indirizzo indicato se: l'indicatore di riporto a 8 bit è attivo; l'indicatore di riporto a 8 bit non è attivo; l'indicatore di zero a 8 bit è attivo; l'indicatore di zero a 8 bit non è attivo; l'indicatore di straripamento a 8 bit è attivo; l'indicatore di straripamento a 8 bit non è attivo; l'indicatore di segno a 8 bit è attivo; l'indicatore di segno a 8 bit non è attivo.
`jump8nc #`***indirizzo***	`01100011`	
`jump8z #`***indirizzo***	`01100100`	
`jump8nz #`***indirizzo***	`01100101`	
`jump8o #`***indirizzo***	`01100110`	
`jump8no #`***indirizzo***	`01100111`	
`jump8n #`***indirizzo***	`01101000`	
`jump8nn #`***indirizzo***	`01101001`	
`jump16c #`***indirizzo***	`01110010`	Salta all'esecuzione dell'istruzione contenuta nell'indirizzo indicato se: l'indicatore di riporto a 16 bit è attivo; l'indicatore di riporto a 16 bit non è attivo; l'indicatore di zero a 16 bit è attivo; l'indicatore di zero a 16 bit non è attivo; l'indicatore di straripamento a 16 bit è attivo; l'indicatore di straripamento a 16 bit non è attivo; l'indicatore di segno a 16 bit è attivo; l'indicatore di segno a 16 bit non è attivo.
`jump16nc #`***indirizzo***	`01110011`	
`jump16z #`***indirizzo***	`01110100`	
`jump16nz #`***indirizzo***	`01110101`	
`jump16o #`***indirizzo***	`01110110`	
`jump16no #`***indirizzo***	`01110111`	
`jump16n #`***indirizzo***	`01111000`	
`jump16nn #`***indirizzo***	`01111001`	
`push8 #`***registro***	`10000`***rrr***	Inserisce nella pila dei dati, oppure recupera dalla pila dei dati, gli otto bit meno significativi del registro indicato, il quale è annotato negli ultimi tre bit del codice operativo. Il caso del registro ***SP*** non è valido, in quanto si tratta dell'indice della pila che non ha senso accantonare.
`pop8 #`***registro***	`10001`***rrr***	
`push16 #`***registro***	`10010`***rrr***	Inserisce nella pila dei dati, oppure recupera dalla pila dei dati, gli otto bit meno significativi del registro indicato, il quale è annotato negli ultimi tre bit del codice operativo. Il caso del registro ***SP*** non è valido, in quanto si tratta dell'indice della pila che non ha senso accantonare.
`pop16 #`***registro***	`10011`***rrr***	
`c81016u`	`10100000`	Estende il contenuto a 8 bit del registro ***A*** in un valore a 16 bit: nel primo caso trattando il valore senza segno, nel secondo trattandolo con segno.
`c81016s`	`10100001`	

Sintassi del macrocodice	Codice operativo binario	Descrizione
`equal`	`10100010`	Operazione logica a partire dal valore dei registri ***A*** e ***B***, mettendo il risultato nel registro ***A***.
`not`	`10100011`	
`and`	`10100100`	
`nand`	`10100101`	
`or`	`10100110`	
`nor`	`10100111`	
`xor`	`101001000`	
`nxor`	`10101001`	
`add`	`10101010`	Esegue l'operazione ***A***+***B***, oppure ***A***−***B***, senza tenere conto del riporto preesistente. Il risultato aggiorna il registro ***A***.
`sub`	`10101011`	
`addc8`	`10101100`	Esegue l'operazione ***A***+***B***, oppure ***A***−***B***, tenendo conto del riporto preesistente e dell'estensione dei valori da sommare o sottrarre. Il risultato aggiorna il registro ***A***.
`subb8`	`10101101`	
`addc16`	`10101110`	
`subb16`	`10101111`	
`lshl8`	`10110000`	Scorrimenti e rotazioni a 8 bit, sul valore del registro ***A***, mettendo il risultato nello stesso registro ***A***.
`lshr8`	`10110001`	
`ashl8`	`10110010`	
`ashr8`	`10110011`	
`rotcl8`	`10110100`	
`rotcr8`	`10110101`	
`rotl8`	`10110110`	
`rotr8`	`10110111`	
`lshl16`	`10111000`	Scorrimenti e rotazioni a 16 bit, sul valore del registro ***A***, mettendo il risultato nello stesso registro ***A***.
`lshr16`	`10111001`	
`ashl16`	`10111010`	
`ashr16`	`10111011`	
`rotcl16`	`10111100`	
`rotcr16`	`10111101`	
`rotl16`	`10111110`	
`rotr16`	`10111111`	
`in #`***io***	`11000000`	Legge o scrive un valore nell'indirizzo di I/O indicato. L'indirizzo di I/O è un numero a 8 bit. Nel caso di '**`is_ack`**', si valuta la presenza di una conferma da parte del dispositivo e, se presente, si salta all'istruzione corrispondente all'indirizzo che appare come ultimo argomento. Attualmente solo il dispositivo dello schermo prevede questo tipo di interrogazione.
`out` ***io***	`11000001`	
`is_ack #`***io***`, #`***indirizzo***	`11000010`	

Sintassi del macrocodice	Codice operativo binario	Descrizione
`clearc` `setc`	`11000100` `11000101`	Azzera o attiva il bit di riporto nel registro ***FL***.
`cmp` `test`	`11000110` `11000111`	Confronta il contenuto di ***A*** e ***B*** simulando una sottrazione o l'operatore logico AND, al solo scopo di aggiornare il registro ***FL***.
`imrl #`***maschera***	`11001000`	Carica la maschera delle interruzioni hardware accettate. L'argomento è da 8 bit, ma valgono solo i 4 bit meno significativi, in quanto esistono solo quattro interruzioni hardware.
`inc %`***registro*** `dec %`***registro***	`11010`***rrr*** `11011`***rrr***	Incrementa o decrementa, di una unità, il registro indicato, il quale si annota negli ultimi tre bit del codice operativo.
`stop`	`11111111`	Ferma la CPU, bloccando il flusso del segnale di clock.

Microcodice

Listato u116.35. Campi in cui si suddividono i bit che rappresentano il microcodice, nelle memorie da ***micro0*** a ***micro4***.

```
field ctrl[1:0]    = {nop=0, stop=1, load=2};
field pc[10:2]     = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field sel[15:11]   = {if_carry_8=1, if_not_carry_8=3,
                      if_zero_8=5, if_not_zero_8=7,
                      if_negative_8=9, if_not_negative_8=11,
                      if_overflow_8=13, if_not_overflow_8=15,
                      if_carry_16=1, if_not_carry_16=3,
                      if_zero_16=5, if_not_zero_16=7,
                      if_negative_16=9, if_not_negative_16=11,
                      if_overflow_16=13, if_not_overflow_16=15};
field mdr[24:16]   = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field i[33:25]     = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field j[42:34]     = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field ram[47:43]   = {br=1, bw=2, aux=4, p=0, i=8, j=16, s=24};
field ivt[50:48]   = {br=1, bw=2, inta=0, intb=4};
field ir[59:51]    = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field bus[77:60]   = {bw=0x10000, aux=0x20000};
field irq[80:78]   = {br=1, bw=2, done=4};
field sp[89:81]    = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field bp[98:90]    = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field b[107:99]    = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field fl[116:108]  = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field alu[127:117] = {bw=1, aux=2, sign=4, rank8=0, rank16=8,
                      a=0, and=16, or=32, xor=48,
                      nxor=64, nor=80, nand=96, not=112,
                      add=256, sub=228, addc=320, subb=352,
                      lshl=512, lshr=528, ashl=484, ashr=560,
                      rotcl=576, rotcr=592,
                      rotl=768, rotr=784,
                      clearc=1024, clearz=1040, clearn=1056,
                      clearo=1072, cleari=1088,
                      setc=1152, setz=1168, setn=1184,
                      seto=1200, seti=1216};
field a[136:128]   = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field ioa[145:137] = {br=1, bw=2, aux=4, low=8, high=16,
                      p1=32, p2=64, m1=128, m2=256};
field ioc[149:146] = {br=1, bw=2, req=4, isack=8};
```


Listato u116.36. Dichiarazione del microcodice.

```
begin microcode @ 0
//
// fetch:
//      IR  <-- RAM[pc++]; load;
//
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
//
nop:
    // fetch:
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
op_error:
    // INT 0
    // push FL
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s fl=bw fl=low  sp=p1;       // RAM[sp++] <- FL[7:0];
    ram=br ram=s fl=bw fl=high sp=m1;       // RAM[sp--] <- FL[15:8];
    // reset interrupt enable flag
    fl=br fl=aux alu=cleari;
    // push PC
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s pc=bw pc=low  sp=p1;       // RAM[sp++] <- PC[7:0];
    ram=br ram=s pc=bw pc=high sp=m1;       // RAM[sp--] <- PC[15:8];
    // push I
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s i=bw i=low  sp=p1;         // RAM[sp++] <- I[7:0];
    ram=br ram=s i=bw i=high sp=m1;         // RAM[sp--] <- I[15:8];
    //
    i=br ivt=bw ivt=intb  bus=bw bus=aux bus=0;     // I <- IVT <- 0;
    pc=br pc=low  ram=bw ram=i i=p1;        // PC[7:0]  <-- RAM[i++]
    pc=br pc=high ram=bw ram=i i=m1;        // PC[15:7] <-- RAM[i--]
    // pop I
    i=br i=low  ram=bw ram=s sp=p1;         // I[7:0] <-- RAM[sp++];
    i=br i=high ram=bw ram=s sp=p1;         // I[15:0] <-- RAM[sp++];
    // fetch
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_j:
    j=br i=bw                               // J <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_a:
    a=br i=bw                               // A <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_b:
    b=br i=bw                               // B <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_bp:
    bp=br i=bw                              // BP <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_sp:
    sp=br i=bw                              // SP <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_mdr:
    mdr=br i=bw                             // MDR <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_i_fl:
    fl=br i=bw                              // FL <- I, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_i:
    i=br j=bw                               // I <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_a:
    a=br j=bw                               // A <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_b:
    b=br j=bw                               // B <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_bp:
    bp=br j=bw                              // BP <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_sp:
    sp=br j=bw                              // SP <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_mdr:
    mdr=br j=bw                             // MDR <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_j_fl:
    fl=br j=bw                              // FL <- J, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_i:
    i=br a=bw                               // I <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_j:
    j=br a=bw                               // J <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_b:
    b=br a=bw                               // B <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_bp:
    bp=br a=bw                              // BP <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_sp:
    sp=br a=bw                              // SP <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_mdr:
    mdr=br a=bw                             // MDR <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_a_fl:
    fl=br a=bw                              // FL <- A, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_i:
```

```
    i=br b=bw                           // I <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_j:
    j=br b=bw                           // J <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_a:
    a=br b=bw                           // A <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_bp:
    bp=br b=bw                          // BP <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_sp:
    sp=br b=bw                          // SP <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_mdr:
    mdr=br b=bw                         // MDR <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_b_fl:
    fl=br b=bw                          // FL <- B, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_i:
    i=br bp=bw                          // I <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_j:
    j=br bp=bw                          // J <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_a:
    a=br bp=bw                          // A <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_b:
    b=br bp=bw                          // B <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_sp:
    sp=br bp=bw                         // SP <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_mdr:
    mdr=br bp=bw                        // MDR <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_bp_fl:
    fl=br bp=bw                         // FL <- BP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_i:
    i=br sp=bw                          // I <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_j:
    j=br sp=bw                          // J <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_a:
    a=br sp=bw                          // A <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_bp:
    bp=br sp=bw                         // BP <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_b:
    b=br sp=bw                          // B <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_mdr:
    mdr=br sp=bw                        // MDR <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_sp_fl:
    fl=br sp=bw                         // FL <- SP, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_i:
    i=br mdr=bw                         // I <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_j:
    j=br mdr=bw                         // J <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_bp:
    bp=br mdr=bw                        // BP <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_sp:
    sp=br mdr=bw                        // SP <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_b:
    b=br mdr=bw                         // B <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_a:
    a=br mdr=bw                         // A <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_mdr_fl:
    fl=br mdr=bw                        // FL <- MDR, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_i:
    i=br fl=bw                          // I <- FL, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_j:
    j=br fl=bw                          // J <- FL, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_a:
    a=br fl=bw                          // A <- FL, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_bp:
    bp=br fl=bw                         // BP <- FL, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_sp:
    sp=br fl=bw                         // SP <- FL, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_mdr:
    mdr=br fl=bw                        // MDR <- FL, fetch
```



```
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
mv_fl_b:
    fl=br b=bw                          // B <- FL, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
load8_i:
    mdr=br ram=bw ram=i;                // MDR <- RAM[i];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
load8_j:
    mdr=br ram=bw ram=j;                // MDR <- RAM[j];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
load8:
    i=br i=low  ram=bw ram=p pc=p1;     // I[7:0] <- RAM[pc++];
    i=br i=high ram=bw ram=p pc=p1;     // I[15:8] <- RAM[pc++];
    mdr=br ram=bw ram=i;                // MDR <- RAM[i];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
load16_i:
    mdr=br mdr=low  ram=bw ram=i i=p1;  // MDR[7:0] <- RAM[i++];
    mdr=br mdr=high ram=bw ram=i i=m1;  // MDR[15:8] <- RAM[i--];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
load16_j:
    mdr=br mdr=low  ram=bw ram=j j=p1;  // MDR[7:0] <- RAM[j++];
    mdr=br mdr=high ram=bw ram=j j=m1;  // MDR[15:8] <- RAM[j--];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
load16:
    i=br i=low  ram=bw ram=p pc=p1;     // I[7:0] <- RAM[pc++];
    i=br i=high ram=bw ram=p pc=p1;     // I[15:8] <- RAM[pc++];
    mdr=br mdr=low  ram=bw ram=i i=p1;  // MDR[7:0] <- RAM[i++];
    mdr=br mdr=high ram=bw ram=i i=m1;  // MDR[15:8] <- RAM[i--];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store8_i:
    ram=br ram=i mdr=bw;                // RAM[i] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store8_j:
    ram=br ram=j mdr=bw;                // RAM[j] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store8:
    i=br i=low  ram=bw ram=p pc=p1;     // I[7:0] <- RAM[pc++];
    i=br i=high ram=bw ram=p pc=p1;     // I[15:8] <- RAM[pc++];
    ram=br ram=i mdr=bw;                // RAM[i] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store16_i:
    ram=br ram=i mdr=bw mdr=low  i=p1;  // RAM[i++] <- MDR[7:0];
    ram=br ram=i mdr=bw mdr=high i=m1;  // RAM[i--] <- MDR[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store16_j:
    ram=br ram=j mdr=bw mdr=low  j=p1;  // RAM[j++] <- MDR[7:0];
    ram=br ram=j mdr=bw mdr=high j=m1;  // RAM[j--] <- MDR[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
store16:
    i=br i=low  ram=bw ram=p pc=p1;     // I[7:0] <- RAM[pc++];
    i=br i=high ram=bw ram=p pc=p1;     // I[15:8] <- RAM[pc++];
    ram=br ram=i mdr=bw mdr=low  i=p1;  // RAM[i++] <- MDR[7:0];
    ram=br ram=i mdr=bw mdr=high i=m1;  // RAM[i--] <- MDR[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
cp8_ij:
    mdr=br ram=bw ram=i i=p1;           // MDR[7:0] <- RAM[i++];
    ram=br ram=j mdr=bw j=p1;           // RAM[j++] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
cp8_ji:
    mdr=br ram=bw ram=j j=p1;           // MDR[7:0] <- RAM[j++];
    ram=br ram=i mdr=bw i=p1;           // RAM[i++] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
cp16_ij:
    mdr=br mdr=low  ram=bw ram=i i=p1;  // MDR[7:0] <- RAM[i++];
    mdr=br mdr=high ram=bw ram=i i=p1;  // MDR[15:8] <- RAM[i++];
    ram=br ram=j mdr=bw mdr=low  j=p1;  // RAM[j++] <- MDR[7:0];
    ram=br ram=j mdr=bw mdr=high j=p1;  // RAM[j++] <- MDR[15:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
cp16_ji:
    mdr=br mdr=low  ram=bw ram=j j=p1;  // MDR[7:0] <- RAM[j++];
    mdr=br mdr=high ram=bw ram=j j=p1;  // MDR[15:8] <- RAM[j++];
    ram=br ram=i mdr=bw mdr=low  i=p1;  // RAM[i++] <- MDR[7:0];
    ram=br ram=i mdr=bw mdr=high i=p1;  // RAM[i++] <- MDR[15:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump:
    i=br pc=bw;                         // I <- PC
    pc=br pc=low  ram=bw ram=i i=p1;    // PC[7:0]  <-- RAM[i++]
    pc=br pc=high ram=bw ram=i i=m1;    // PC[15:7] <-- RAM[i--]
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
jump8c:
    mdr=br mdr=low  ram=bw ram=p pc=p1; // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1; // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_carry_8;               // PC = (carry8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8nc:
    mdr=br mdr=low  ram=bw ram=p pc=p1; // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1; // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_carry_8;           // PC = (not_carry8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8z:
    mdr=br mdr=low  ram=bw ram=p pc=p1; // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1; // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_zero_8;                // PC = (zero8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8nz:
    mdr=br mdr=low  ram=bw ram=p pc=p1; // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1; // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_zero_8;            // PC = (not_zero8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8o:
```

```
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_overflow_8;                // PC = (overflow8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8no:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_overflow_8;            // PC = (not_overflow8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8n:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_negative_8;                // PC = (negative8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump8nn:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_negative_8;            // PC = (not_negative8?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16c:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_carry_16;                  // PC = (carry16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16nc:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_carry_16;              // PC = (not_carry16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16z:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_zero_16;                   // PC = (zero16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16nz:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_zero_16;               // PC = (not_zero16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16o:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_overflow_16;               // PC = (overflow16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16no:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_overflow_16;           // PC = (not_overflow16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16n:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_negative_16;               // PC = (negative16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
jump16nn:
    mdr=br mdr=low  ram=bw ram=p pc=p1;     // MDR[7:0] <-- RAM[pc++]
    mdr=br mdr=high ram=bw ram=p pc=p1;     // MDR[15:7] <-- RAM[pc++]
    pc=br sel=if_not_negative_16;           // PC = (not_negative16?MDR:PC)
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
call:
    i=br i=low  ram=bw ram=p pc=p1 sp=m1;   // I[7:0] <-- RAM[pc++], SP--
    i=br i=high ram=bw ram=p pc=p1 sp=m1;   // I[15:7] <-- RAM[pc++], SP--
    ram=br ram=s pc=bw pc=low  sp=p1;       // RAM[sp++] <- PC[7:0], SP++
    ram=br ram=s pc=bw pc=high sp=m1;       // RAM[sp--] <- PC[15:8], SP--
    pc=br i=bw;                             // PC <- I;
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
call_i:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s pc=bw pc=low  sp=p1;       // RAM[sp++] <- PC[7:0], SP++
    ram=br ram=s pc=bw pc=high sp=m1;       // RAM[sp--] <- PC[15:8], SP--
    pc=br i=bw;                             // PC <- I;
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
call_j:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s pc=bw pc=low  sp=p1;       // RAM[sp++] <- PC[7:0], SP++
    ram=br ram=s pc=bw pc=high sp=m1;       // RAM[sp--] <- PC[15:8], SP--
    pc=br j=bw;                             // PC <- J;
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
return:
    pc=br pc=low  ram=bw ram=s sp=p1;       // PC[7:0] <- RAM[sp++];
    pc=br pc=high ram=bw ram=s sp=p1;       // PC[15:8] <- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_i:
    sp=m1;                                  // SP--;
    ram=br ram=s i=bw i=low;                // RAM[sp] <- I[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_j:
    sp=m1;                                  // SP--;
    ram=br ram=s j=bw j=low;                // RAM[sp] <- J[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_a:
    sp=m1;                                  // SP--;
    ram=br ram=s a=bw a=low;                // RAM[sp] <- A[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_b:
    sp=m1;                                  // SP--;
    ram=br ram=s b=bw b=low;                // RAM[sp] <- B[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_bp:
    sp=m1;                                  // SP--;
```



```
    ram=br ram=s bp=bw bp=low;              // RAM[sp] <- BP[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_mdr:
    sp=m1;                                  // SP--;
    ram=br ram=s mdr=bw mdr=low;            // RAM[sp] <- MDR[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push8_fl:
    sp=m1;                                  // SP--;
    ram=br ram=s fl=bw fl=low;              // RAM[sp] <- FL[7:0];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_i:
    i=br i=low  ram=bw ram=s sp=p1          // I[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_j:
    j=br j=low  ram=bw ram=s sp=p1;         // J[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_a:
    a=br a=low  ram=bw ram=s sp=p1;         // A[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_b:
    b=br b=low  ram=bw ram=s sp=p1;         // B[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_bp:
    bp=br bp=low  ram=bw ram=s sp=p1;       // BP[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_mdr:
    mdr=br mdr=low  ram=bw ram=s sp=p1;     // MDR[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop8_fl:
    fl=br fl=low  ram=bw ram=s sp=p1;       // FL[7:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_i:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s i=bw i=low  sp=p1;         // RAM[sp++] <- I[7:0];
    ram=br ram=s i=bw i=high sp=m1;         // RAM[sp--] <- I[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_j:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s j=bw j=low  sp=p1;         // RAM[sp++] <- J[7:0];
    ram=br ram=s j=bw j=high sp=m1;         // RAM[sp--] <- J[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_a:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s a=bw a=low  sp=p1;         // RAM[sp++] <- A[7:0];
    ram=br ram=s a=bw a=high sp=m1;         // RAM[sp--] <- A[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_b:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s b=bw b=low  sp=p1;         // RAM[sp++] <- B[7:0];
    ram=br ram=s b=bw b=high sp=m1;         // RAM[sp--] <- B[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_bp:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s bp=bw bp=low  sp=p1;       // RAM[sp++] <- BP[7:0];
    ram=br ram=s bp=bw bp=high sp=m1;       // RAM[sp--] <- BP[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_mdr:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s mdr=bw mdr=low  sp=p1;     // RAM[sp++] <- MDR[7:0];
    ram=br ram=s mdr=bw mdr=high sp=m1;     // RAM[sp--] <- MDR[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
push16_fl:
    sp=m2;                                  // SP <-- (SP - 2)
    ram=br ram=s fl=bw fl=low  sp=p1;       // RAM[sp++] <- FL[7:0];
    ram=br ram=s fl=bw fl=high sp=m1;       // RAM[sp--] <- FL[15:8];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_i:
    i=br i=low  ram=bw ram=s sp=p1;         // I[7:0] <-- RAM[sp++];
    i=br i=high ram=bw ram=s sp=p1;         // I[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_j:
    j=br j=low  ram=bw ram=s sp=p1;         // J[7:0] <-- RAM[sp++];
    j=br j=high ram=bw ram=s sp=p1;         // J[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_a:
    a=br a=low  ram=bw ram=s sp=p1;         // A[7:0] <-- RAM[sp++];
    a=br a=high ram=bw ram=s sp=p1;         // A[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_b:
    b=br b=low  ram=bw ram=s sp=p1;         // B[7:0] <-- RAM[sp++];
    b=br b=high ram=bw ram=s sp=p1;         // B[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_bp:
    bp=br bp=low  ram=bw ram=s sp=p1;       // BP[7:0] <-- RAM[sp++];
    bp=br bp=high ram=bw ram=s sp=p1;       // BP[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_mdr:
    mdr=br mdr=low  ram=bw ram=s sp=p1;     // MDR[7:0] <-- RAM[sp++];
    mdr=br mdr=high ram=bw ram=s sp=p1;     // MDR[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
pop16_fl:
    fl=br fl=low  ram=bw ram=s sp=p1;       // FL[7:0] <-- RAM[sp++];
    fl=br fl=high ram=bw ram=s sp=p1;       // FL[15:0] <-- RAM[sp++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
c8to16u:
    a=br alu=bw alu=a alu=rank8 fl=br fl=aux        // A[15:0] <- A[7:0],
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;    // fetch;
c8to16s:
    a=br alu=bw alu=a alu=rank8 alu=sign fl=br fl=aux   // A[15:0] <- A[7:0],
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;    // fetch;
```

```
equal:
    a=br alu=bw alu=a alu=rank16 fl=br fl=aux       // A <- A, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
not:
    a=br alu=bw alu=not alu=rank16 fl=br fl=aux     // A <- NOT A, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
and:
    a=br alu=bw alu=and alu=rank16 fl=br fl=aux     // A <- A AND B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
nand:
    a=br alu=bw alu=nand alu=rank16 fl=br fl=aux    // A <- A NAND B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
or:
    a=br alu=bw alu=or alu=rank16 fl=br fl=aux      // A <- A OR B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
nor:
    a=br alu=bw alu=nor alu=rank16 fl=br fl=aux     // A <- A NOR B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
xor:
    a=br alu=bw alu=xor alu=rank16 fl=br fl=aux     // A <- A XOR B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
nxor:
    a=br alu=bw alu=nxor alu=rank16 fl=br fl=aux    // A <- A NXOR B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
add:
    a=br alu=bw alu=add alu=rank16 fl=br fl=aux     // A <- A+B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
sub:
    a=br alu=bw alu=sub alu=rank16 fl=br fl=aux     // A <- A-B, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
addc8:
    a=br alu=bw alu=addc alu=rank8 fl=br fl=aux     // A <- A+B+carry, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
subb8:
    a=br alu=bw alu=subb alu=rank8 fl=br fl=aux     // A <- A-B-borrow, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
addc16:
    a=br alu=bw alu=addc alu=rank16 fl=br fl=aux    // A <- A+B+carry, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
subb16:
    a=br alu=bw alu=subb alu=rank16 fl=br fl=aux    // A <- A-B-borrow, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
lshl8:
    a=br alu=bw alu=lshl alu=rank8 fl=br fl=aux    // A <- A <<, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
lshr8:
    a=br alu=bw alu=lshr alu=rank8 fl=br fl=aux    // A <- A >>, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
ashl8:
    a=br alu=bw alu=ashl alu=rank8 alu=sign fl=br fl=aux    // A <- A*2,
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;    // fetch:
ashr8:
    a=br alu=bw alu=ashr alu=rank8 alu=sign fl=br fl=aux   // A <- A/2, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotcl8:
    a=br alu=bw alu=rotcl alu=rank8 fl=br fl=aux   // A <- rotcl(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotcr8:
    a=br alu=bw alu=rotcr alu=rank8 fl=br fl=aux   // A <- rotcr(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotl8:
    a=br alu=bw alu=rotl alu=rank8 fl=br fl=aux    // A <- rotl(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotr8:
    a=br alu=bw alu=rotr alu=rank8 fl=br fl=aux    // A <- rotr(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
lshl16:
    a=br alu=bw alu=lshl alu=rank16 fl=br fl=aux   // A <- A <<, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
lshr16:
    a=br alu=bw alu=lshr alu=rank16 fl=br fl=aux   // A <- A >>, fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
ashl16:
    a=br alu=bw alu=ashl alu=rank16 alu=sign fl=br fl=aux   // A <- A*2,
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;    // fetch;
ashr16:
    a=br alu=bw alu=ashr alu=rank16 alu=sign fl=br fl=aux   // A <- A/2,
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;    // fetch;
rotcl16:
    a=br alu=bw alu=rotcl alu=rank16 fl=br fl=aux  // A <- rotcl(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotcr16:
    a=br alu=bw alu=rotcr alu=rank16 fl=br fl=aux  // A <- rotcr(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotl16:
    a=br alu=bw alu=rotl alu=rank16 fl=br fl=aux   // A <- rotl(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
rotr16:
    a=br alu=bw alu=rotr alu=rank16 fl=br fl=aux   // A <- rotr(A), fetch
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
in:
    ioa=br ram=bw ram=p pc=p1;                  // IOA <- RAM[pc++];
    ioc=req;                                    // I/O request;
    ctrl=nop;                                   // non fa alcunché
    a=br ioc=bw                                 // A <- I/O, fetch;
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
out:
    ioa=br ram=bw ram=p pc=p1;                  // IOA <- RAM[pc++];
    ioc=br a=bw;                                // I/O <- A
    ioc=req                                     // I/O request, fetch;
```



```
      ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
is_ack:
    ioa=br ram=bw ram=p pc=p1;                  // IOA <- RAM[pc++];
    mdr=br mdr=low  ram=bw ram=p pc=p1;         // MDR[7:0] <- RAM[pc++];
    mdr=br mdr=high ram=bw ram=p pc=p1;         // MDR[15:8] <- RAM[pc++];
    a=br ioc=bw ioc=isack;                      // A <- I/O is ack;
    a=br alu=a alu=rank8 alu=sign fl=br fl=aux; // A[15:0] <- A[7:0];
    pc=br sel=if_not_zero_8;                    // PC = (not_zero8?MDR:PC);
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
int:
    // push FL
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s fl=bw fl=low  sp=p1;           // RAM[sp++] <- FL[7:0];
    ram=br ram=s fl=bw fl=high sp=m1;           // RAM[sp--] <- FL[15:8];
    // reset interrupt enable flag, pc++
    // (PC viene incrementato per saltare l'argomento, prima di
    // salvare il suo valore nella pila).
    fl=br fl=aux alu=cleari pc=p1;
    // push PC
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s pc=bw pc=low  sp=p1;           // RAM[sp++] <- PC[7:0];
    ram=br ram=s pc=bw pc=high sp=m1;           // RAM[sp--] <- PC[15:8];
    // push I
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s i=bw i=low  sp=p1;             // RAM[sp++] <- I[7:0];
    ram=br ram=s i=bw i=high sp=m1;             // RAM[sp--] <- I[15:8];
    // riporta PC al valore corretto per individuare
    // l'argomento che contiene il numero di interruzione.
    pc=m1;
    //
    i=br ivt=bw ivt=intb ram=bw ram=aux ram=p pc=p1; // I <- IVT <- RAM[pc++];
    pc=br pc=low  ram=bw ram=i i=p1;            // PC[7:0]  <-- RAM[i++]
    pc=br pc=high ram=bw ram=i i=m1;            // PC[15:7] <-- RAM[i--]
    // pop I
    i=br i=low  ram=bw ram=s sp=p1;             // I[7:0] <-- RAM[sp++];
    i=br i=high ram=bw ram=s sp=p1;             // I[15:0] <-- RAM[sp++];
    // fetch
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
iret:
    // pop PC
    pc=br pc=low  ram=bw ram=s sp=p1;           // PC[7:0] <- RAM[sp++];
    pc=br pc=high ram=bw ram=s sp=p1;           // PC[15:8] <- RAM[sp++];
    // pop FL
    fl=br fl=low  ram=bw ram=s sp=p1;           // FL[7:0] <-- RAM[sp++];
    fl=br fl=high ram=bw ram=s sp=p1;           // FL[15:0] <-- RAM[sp++];
    // fetch
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
irq:
    // push FL
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s fl=bw fl=low  sp=p1;           // RAM[sp++] <- FL[7:0];
    ram=br ram=s fl=bw fl=high sp=m1;           // RAM[sp--] <- FL[15:8];
    // reset interrupt enable flag
    fl=br fl=aux alu=cleari;
    // ripristina il valore corretto di PC:
    // PC è collocato dopo il codice operativo
    // di un'istruzione al posto della quale si sta
    // eseguendo il codice dell'interruzione; pertanto,
    // il valore corretto di PC da salvare è PC-1.
    pc=m1;                                      // PC--;
    // push PC
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s pc=bw pc=low  sp=p1;           // RAM[sp++] <- PC[7:0];
    ram=br ram=s pc=bw pc=high sp=m1;           // RAM[sp--] <- PC[15:8];
    // push I
    sp=m2;                                      // SP <-- (SP - 2)
    ram=br ram=s i=bw i=low  sp=p1;             // RAM[sp++] <- I[7:0];
    ram=br ram=s i=bw i=high sp=m1;             // RAM[sp--] <- I[15:8];
    //.
    i=br ivt=bw ivt=inta;                       // I <- IVT <- IRQ;
    pc=br pc=low  ram=bw ram=i i=p1;            // PC[7:0]  <-- RAM[i++]
    pc=br pc=high ram=bw ram=i i=m1;            // PC[15:7] <-- RAM[i--]
    // pop I
    i=br i=low  ram=bw ram=s sp=p1;             // I[7:0] <-- RAM[sp++];
    i=br i=high ram=bw ram=s sp=p1;             // I[15:0] <-- RAM[sp++];
    //
    irq=done;
    // fetch
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
ivtl:
    i=br i=low  ram=bw ram=p pc=p1;             // I[7:0] <- RAM[pc++];
    i=br i=high ram=bw ram=p pc=p1;             // I[15:8] <- RAM[pc++];
    ivt=br i=bw;                                // IVT <- MDR;
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
imrl:
    irq=br ram=bw ram=p pc=p1;                  // IRQ <- RAM[pc++];
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;  // fetch
cleari:
    fl=br fl=aux alu=cleari irq=done
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
seti:
    fl=br fl=aux alu=seti irq=done
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
clearc:
    fl=br fl=aux alu=clearc
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
setc:
    fl=br fl=aux alu=setc
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
cmp:
```

```
    fl=br fl=aux alu=sub                    // FL(A - B);
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
test:
    fl=br fl=aux alu=and                    // FL(A AND B);
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_i:
    i=p1                                    // I++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_j:
    j=p1                                    // J++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_a:
    a=p1                                    // A++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_b:
    b=p1                                    // B++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_bp:
    bp=p1                                   // BP++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_sp:
    sp=p1                                   // SP++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_mdr:
    mdr=p1                                  // MDR++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
inc_fl:
    fl=p1                                   // FL++, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_i:
    i=m1                                    // I--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_j:
    j=m1                                    // J--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_a:
    a=m1                                    // A--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_b:
    b=m1                                    // B--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_bp:
    bp=m1                                   // BP--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_sp:
    sp=m1                                   // SP--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_mdr:
    mdr=m1                                  // MDR--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
dec_fl:
    fl=m1                                   // FL--, fetch;
        ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
//
stop:
    ctrl=stop;                                  // stop clock
    // if resumed, fetch:
    ir=aux ir=br ram=aux ram=bw ram=p pc=p1 ctrl=load;
end
```

Gestione delle interruzioni

Si distinguono tre tipi di interruzioni: quelle generate internamente dalla CPU, quelle hardware (IRQ) e quelle software. Le interruzioni interne di CPU vanno da INT0 a INT3, ma attualmente è previsto solo INT0 che corrisponde all'individuazione di un codice operativo non valido. Le interruzioni hardware vanno da INT4 a INT7 e corrispondono rispettivamente all'intervallo da IRQ0 a IRQ3. Le interruzioni software vanno da INT8 a INT15. La tabella IVT (*interrupt vector table*) va predisposta nel macrocodice e va inizializzato il registro ***IVT*** con l'indirizzo della sua collocazione, come nell'esempio seguente:

```
begin macrocode @ 0
        jump #start
        nop
interrupt_vector_table:
        .short  0x0025  // CPU # op_code_error
        .short  0x0024  // CPU # default_interrupt_routine
        .short  0x0024  // CPU # default_interrupt_routine
        .short  0x0024  // CPU # default_interrupt_routine
        .short  0x0024  // IRQ # default_interrupt_routine
        .short  0x0024  // IRQ # default_interrupt_routine
        .short  0x0024  // IRQ # default_interrupt_routine
        .short  0x0024  // IRQ # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
```



```
        .short  0x0024  // software # default_interrupt_routine
        .short  0x0024  // software # default_interrupt_routine
default_interrupt_routine:
        iret
op_code_error:
        stop
start:
        load16 #sp_base
        mv %MDR, %SP
        ivtl #interrupt_vector_table
        imrl 0x0F // tutti
        seti
        …
        …
sp_base:
        .short 0x0080
end
```

Eventualmente, la tabella può essere più breve, se non si vogliono utilizzare le interruzioni software.

Il manifestarsi di un'interruzione (di CPU, hardware o software) comporta il salvataggio nella pila dei dati del registro ***FL*** (azzerando subito dopo l'indicatore di abilitazione delle interruzioni hardware) e del registro ***PC***, che l'istruzione **`iret`** va poi a recuperare. Ma mentre la conclusione di un'interruzione avviene sempre nello stesso modo, attraverso la descrizione del codice operativo **`iret`**, l'inizio è diverso nei tre casi. Se si tratta di un'interruzione dovuta a un codice operativo errato, viene eseguito il microcodice a partire dall'etichetta '**`op_error:`**'; se si tratta di un'interruzione hardware, viene eseguito il microcodice a partire dall'etichetta '**`irq:`**', quando l'unità di controllo starebbe per passare all'esecuzione di un nuovo codice operativo, ma viene invece dirottata a causa dell'interruzione; se si tratta di un'interruzione software, viene eseguito il microcodice a partire dall'etichetta '**`int:`**', corrispondente al codice operativo **`int`**. Le tre situazioni sono diverse:

- L'interruzione dovuta a un codice operativo errato, comporta un errore nel codice contenuto nella memoria RAM e non si può conoscere l'entità di questo danno. Non potendo fare ipotesi, la scelta migliore per la routine associata all'interruzione dovrebbe coincidere con l'arresto della CPU; diversamente, se si accetta di ritornare all'esecuzione del codice si passa a quanto contenuto nella cella di memoria successiva, senza poter sapere se lì si trova eventualmente un argomento (errato) per il codice operativo errato precedente.
- L'interruzione dovuta a un IRQ avviene in modo asincrono rispetto all'attività della CPU e viene servita quando la CPU stessa starebbe invece per acquisire un nuovo codice operativo. In questa condizione, il registro ***PC*** punta già alla posizione di memoria successiva al codice operativo che avrebbe dovuto essere eseguito; pertanto, prima di salvare il registro ***PC*** nella pila dei dati, occorre farlo arretrare di una posizione, in modo che corrisponda alla posizione del codice operativo che deve essere eseguito al termine dell'interruzione.

  Il microcodice che serve un'interruzione hardware ha anche il compito, una volta letto l'indirizzo corrispondente alla cella della tabella IVT corrispondente, di cancellare la richiesta nel modulo **`IRQ`**. Ciò avviene inviando un segnale tramite il bus di controllo, che nel modulo **`IRQ`** viene recepito come *irq done*. È poi compito della logica del modulo **`IRQ`** sapere qual è effettivamente il segnale di IRQ da azzerare. Contestualmente, il modulo **`IRQ`** potrebbe richiedere la gestione di un'altra interruzione, ma temporaneamente tale gestione risulterebbe sospesa, perché l'indicatore di abilitazione delle interruzioni hardware si trova sicuramente a essere disabilitato (ciò avviene subito dopo il salvataggio del registro ***FL*** nella pila dei dati).
- L'interruzione software è più semplice da governare, perché avviene in modo prevedibile, senza interrompere veramente l'attività dell'unità di controllo.

Quando si verifica un'interruzione esiste anche la necessità di saltare correttamente alla routine prevista nella tabella IVT. Il registro ***IVT*** ha due ingressi distinti per ricevere il numero di interruzione da convertire in indirizzo di memoria: uno è collegato al bus ausiliario, a cui è collegato anche il modulo **`bus`** e il modulo della memoria RAM; l'altro è collegato a modulo **`IRQ`**. Quando si tratta di un'interruzione interna di CPU, il modulo **`IVT`** viene pilotato direttamente dall'unità di controllo, attraverso il modulo **`bus`**; quando si tratta di un'interruzione hardware, il modulo **`IVT`** viene pilotato dal modulo **`IRQ`**; quando invece si tratta di un'interruzione software, il modulo **`IVT`** viene pilotato dalla RAM, dalla quale si preleva il numero dell'interruzione, fornito in qualità di argomento del codice operativo **`int`**. A questo proposito va anche osservato che con il codice operativo **`int`** è possibile attivare qualunque tipo di interruzione, anche se non sarebbe di competenza del software.

Orologio: modulo «RTC»

Il modulo **`RTC`** (*real time clock*) produce un impulso al secondo e si limita a fornirlo attraverso l'interruzione hardware IRQ0. Se nel modulo **`IRQ`** risulta abilitata questa linea di interruzione, a ogni secondo viene richiesta l'interruzione saltando all'indirizzo contenuto nella quinta posizione della tabella IVT; in pratica, IRQ0 corrisponde a INT4 nella tabella IVT.

Il modulo **`RTC`** è costruito semplicemente attraverso codice Verilog:

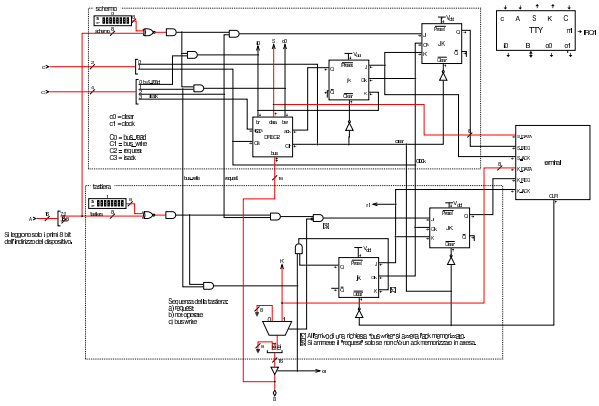
Listato u116.38. Dichiarazione del modulo **`RTC`**.

```
module RTC (T);
  output T;
  reg p;
  always
    begin
      p = 0;
      $tkg$wait (500);
      p = 1;
      $tkg$wait (500);
    end
  assign T = p;
endmodule
```

Modulo «TTY»

Il modulo **`TTY`**, per la gestione del terminale video-tastiera, è quasi identico alla versione precedente della CPU dimostrativa: si aggiunge un'uscita collegata al segnale di conferma (*acknowledge*) della tastiera, per pilotare il segnale IRQ1. In tal modo, quando si preme un tasto sulla tastiera si produce anche un'interruzione IRQ1, la quale può servire per eseguire il codice necessario a prelevare quanto digitato.

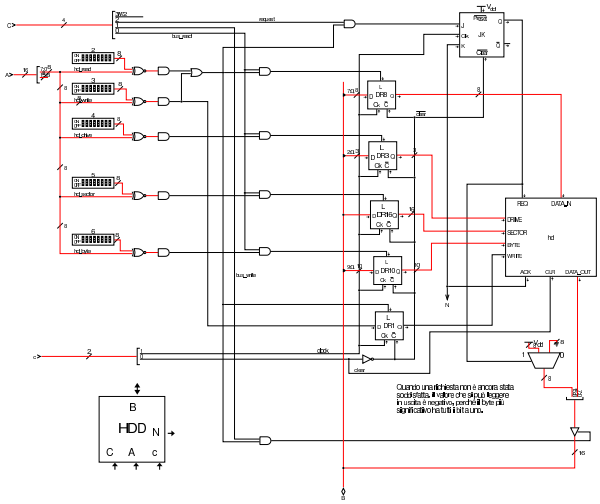
Figura u116.39. Modulo **`TTY`**.



Modulo «HDD»

Il modulo **`HDD`** è nuovo rispetto alla versione precedente: si tratta di un'interfaccia che simula un insieme di otto unità di memorizzazione di massa, suddivise a loro volta in settori da 512 byte ognuno. Al dispositivo si accede con indirizzi di I/O differenti, a seconda del tipo di operazione che si deve svolgere.

Figura u116.40. Schema del modulo **`HDD`**.



Listato u116.41. Codice Verilog che descrive il modulo **hd**.

```
module hd(DRIVE, SECTOR, BYTE, WRITE, DATA_IN, DATA_OUT, REQ, ACK, CLR);
input [2:0] DRIVE;
input WRITE, REQ, CLR;
input [15:0] SECTOR;
input [9:0] BYTE;
input [7:0] DATA_IN;
output [7:0] DATA_OUT;
output ACK;
//
integer _data_out;
integer _ack;
//
reg [7:0] buffer[0:1023];
reg [8+24-1:0] filename = "hd0_sector_000000000.mem";
//
integer i;
integer sector_8;
integer sector_7;
integer sector_6;
integer sector_5;
integer sector_4;
integer sector_3;
integer sector_2;
integer sector_1;
integer sector_0;
integer x;
//
  initial
    begin
      for (i=0; i<1024; i=i+1)
        begin
          //
          // Initial buffer reset with 00.
          //
          buffer[i] = 8'h00;
        end
      _ack = 0;
      _data_out = 0;
      x = 0;
    end
  //
  always
    begin
      @(posedge CLR)
      _ack = 0;
      _data_out = 0;
      x = 0;
    end
  //
  //
  //
  always
    begin
      //
      // Start after a positive edge from REQ!.
```

```
      //
      @(posedge REQ);
      # 10;
      //
      // Define the sector file name.
      //
      x = SECTOR;
      sector_0 = x%10;
      x = x/10;
      sector_1 = x%10;
      x = x/10;
      sector_2 = x%10;
      x = x/10;
      sector_3 = x%10;
      x = x/10;
      sector_4 = x%10;
      x = x/10;
      sector_5 = x%10;
      x = x/10;
      sector_6 = x%10;
      x = x/10;
      sector_7 = x%10;
      x = x/10;
      sector_8 = x%10;
      //
      // La stringa parte da destra verso sinistra!
      //
      filename[12*8+7:12*8] = sector_8 + 8'd48;
      filename[11*8+7:11*8] = sector_7 + 8'd48;
      filename[10*8+7:10*8] = sector_6 + 8'd48;
      filename[9*8+7:9*8] = sector_5 + 8'd48;
      filename[8*8+7:8*8] = sector_4 + 8'd48;
      filename[7*8+7:7*8] = sector_3 + 8'd48;
      filename[6*8+7:6*8] = sector_2 + 8'd48;
      filename[5*8+7:5*8] = sector_1 + 8'd48;
      filename[4*8+7:4*8] = sector_0 + 8'd48;
      //
      filename[21*8+7:21*8] = DRIVE + 8'd48;
      //
      if (WRITE)
        begin
          //
          // Put data inside the buffer.
          //
          buffer[BYTE] = DATA_IN;
          //
          // Save the buffer to disk.
          // Please remember that $writememh() must be enabled inside
          // Tkgate configuration!
          //
          $writememh(filename, buffer);
          //
          // Return the same data read.
          //
          _data_out = buffer[BYTE];
        end
      else
        begin
          //
          // Get data from disk to the buffer.
          //
          $readmemh(filename, buffer);
          //
          // Return the data required.
          //
          _data_out = buffer[BYTE];
        end
      //
      // Acknowledge.
      //
      _ack = 1;
      //
      // Wait the end of request (the negative edge)
      // before restarting the loop.
      //
      @(negedge REQ);
      # 10;
      //
      // Now become ready again.
      //
      _ack = 0;
    end
    //
  assign DATA_OUT = _data_out;
  assign ACK = _ack;
  //
endmodule
```

Trattandosi di un modulo nuovo, è necessario descrivere prima il comportamento di **hd**, di cui è appena stato mostrato il sorgente Verilog: gli ingressi ***DRIVE***, ***SECTOR*** e ***BYTE*** servono a individuare in modo univoco un certo byte, appartenente a un certo settore di una certa unità di memorizzazione. In pratica, ogni unità di memorizzazione virtuale è divisa in settori, dal primo, corrispondente a zero, all'ultimo, corrispondente a 65536. Dal momento che ogni settore è da 512 byte, queste unità di memorizzazione virtuali hanno una capacità massima di 32 Mibyte.

L'ingresso ***WRITE*** consente di selezionare un accesso in scrittura al dispositivo di memorizzazione, altrimenti si intende un accesso in lettura. L'accesso all'unità avviene un byte alla volta e si deve utilizzare l'uscita ***DATA_OUT*** per la lettura, oppure l'ingresso ***DATA_IN*** per la scrittura. Gli ingressi e le uscite ***REQ***, ***ACK*** e ***CLR*** funzionano in modo prevedibile, conformemente a quanto già visto a proposito del dispositivo del terminale (tastiera e schermo).

Per poter usare il dispositivo **HDD**, è necessario fornire inizialmente le coordinate del byte a cui si è interessati, scrivendo nelle porte di I/O 4, 5 e 6, rispettivamente per l'unità di memorizzazione, il settore e il byte. Quindi si può chiedere un'operazione di lettura (indirizzo di I/O 2) o di scrittura (indirizzo di I/O 3). Quando un'operazione di lettura o scrittura è stata completata, il segnale di conferma (*acknowledge*) viene emesso dal modulo **HDD** e diretto al modulo **IRQ**, diventando un segnale IRQ2. Tuttavia si può fare a meno di usare le interruzioni con il modulo **HDD**, perché la lettura è sempre possibile, con la differenza che se il dato ottenuto non è ancora valido, il valore letto è negativo. Allo stesso modo, dopo la scrittura si può verificare che l'operazione sia stata completata attraverso una lettura: se il valore che si ottiene fosse negativo, significherebbe che occorre attendere ancora un po'.

Il modulo **hd** permette di usare i dispositivi di memorizzazione virtuali in modo libero, senza bisogno di creare prima dei file: quando si accede per la prima volta, in scrittura, a un settore che non era mai stato usato prima, viene creato al volo il file che lo rappresenta, nella directory in cui sta lavorando Tkgate. Se invece si legge un settore che non esiste, il dispositivo si limita a produrre il valore nullo. I file che vengono creati corrispondono al modello '`hd`***`n`***`_sector_`***`ssssssss`***`.mem`'.

Macrocodice: esempio di uso del terminale con le interruzioni

Il codice seguente esegue la lettura della tastiera, attraverso l'interruzione generata dalla stessa, e la rappresentazione del testo digitato attraverso lo schermo. La parte iniziale del codice definisce la collocazione della tabella IVT e del codice associato ai suoi vari elementi.

Listato u116.42. Macrocodice per la gestione del terminale attraverso le interruzioni della tastiera.

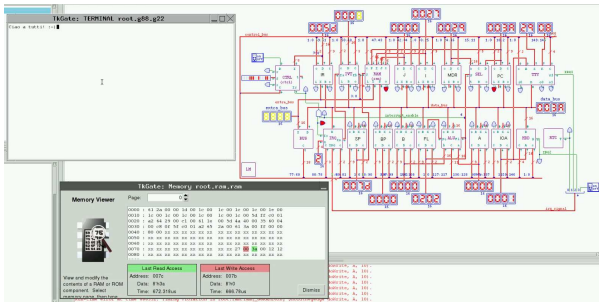
```
begin macrocode @ 0
        jump #start
        nop
interrupt_vector_table:
        .short  0x001D  // CPU
        .short  0x001C  // CPU
        .short  0x001C  // CPU
        .short  0x001C  // CPU

        .short  0x001C  // IRQ
        .short  0x001E  // IRQ keyboard
        .short  0x001C  // IRQ
        .short  0x001C  // IRQ

        .short  0x001C  // software
        .short  0x001C  // software
        .short  0x001C  // software
        .short  0x001C  // software
default_interrupt_routine:
        iret
op_code_error:
        stop
keyboard:
        in 1            // Legge dalla tastiera.
        equal
        jump8z #keyboard_end
        out 0           // Altrimenti emette lo stesso
                        //   valore sullo schermo.
```

```
        jump #keyboard
keyboard_end:
        iret
start:
        load16 #data_1
        mv %MDR, %SP
        //
        ivtl #interrupt_vector_table
        imrl 0x0F // tutti
        seti
keyboard_reset:
        in 1            // Legge dalla tastiera.
        equal
        jump8nz #start // Ripete fino a svuotare il buffer
ciclo:
        jump #ciclo
stop:                   // Non raggiunge mai questo punto.
        stop
data_0:
        .short 0
data_1:
        .short 0x0080
end
```

Figura u116.43. Inserimento da tastiera e visualizzazione sullo schermo. Video: *http://www.youtube.com/watch?v=dgIfZHNTedM*



Riferimenti

- *Tkgate*, *http://www.tkgate.org*
- Albert Paul Malvino, Jerard A. Brown, *Digital Computer Electronics*, Glencoe/Mcgraw-Hill *http://www.amazon.it/Digital-Computer-Electronics-Albert-Malvino/dp/0028005945*
- Olivier Carton, *Circuits et architecture des ordinateurs*, *http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/archi.pdf*

Parte xxvii

Pascal

Pascal: preparazione di alcuni compilatori comuni 953

Pascal-to-C 953
GPC 956
Free-Pascal 957

Pascal: introduzione 959

Struttura fondamentale 959
Variabili e tipi 961
Operatori ed espressioni 962
Strutture di controllo del flusso 963
Procedure e funzioni 967
I/O elementare 970
Struttura del sorgente: le dichiarazioni 971
Riferimenti 972

Pascal: tipi di dati derivati 973

Array 973
Stringhe 974
Tipi 975
Costanti 975
Tipo enumerativo, sottointervallo e insieme 976
Record 977
Riferimenti 978

Pascal: esempi di programmazione 979

Problemi elementari di programmazione 979
Scansione di array 985
Algoritmi tradizionali 987

Pascal: preparazione di alcuni compilatori comuni

Pascal-to-C 953
Librerie e compilazione 953
Uso di Pascal-to-C 955
GPC 956
Free-Pascal 957

Sono disponibili diversi compilatori per il linguaggio Pascal; in questo capitolo ne vengono descritti alcuni.

Pascal-to-C

Pascal-to-C,[1] è una sorta di compilatore che permette di convertire un sorgente Pascal in un sorgente C. I problemi che possono sorgere da questo tipo di conversione sono nella definizione precisa del tipo di dialetto Pascal e del tipo di dialetto C. Utilizzando Pascal-to-C con GNU/Linux, non si dovrebbero avere difficoltà con il compilatore C. Quello che resta da sistemare è la definizione del dialetto Pascal che si vuole usare, dal momento che ne esistono di diversi, che alle volte sono incompatibili.

Questi dettagli possono essere controllati e configurati; quello che conta è esserne consapevoli e approfondire l'uso di Pascal-to-C attraverso lo studio della documentazione originale, quando se ne presenta la necessità, ovvero quando si intende programmare seriamente attraverso questo strumento.

Il nome di Pascal-to-C è indicato dal suo autore come P2c. Tuttavia, P2C è anche il nome di un altro compilatore analogo, realizzato per sistemi speciali: *http://www.geocities.com/SiliconValley/Network/3656/rocket/* . In questo secondo caso, oltre alla particolarità del compilatore stesso, c'è da considerare il fatto che non si tratta di software libero.

Librerie e compilazione

Il codice C generato da Pascal-to-C contiene sempre l'inclusione del file '`p2c/p2c.h`', che poi, a sua volta, provvede a includere il solito '`stdio.h`'.

Il *link* del file generato dalla compilazione del sorgente C che si ottiene, deve essere fatto includendo la libreria '`libp2c.a`', cosa che si traduce generalmente nell'uso dell'opzione '**`-lp2c`**'.

In pratica, le fasi necessarie a ottenere un programma eseguibile si riassumono nei due comandi seguenti.

```
p2c sorgente_pascal
```

```
cc -lp2c sorgente_c
```

L'eseguibile che si ottiene, richiede la presenza della libreria dinamica '`libp2c.so`'.

Il funzionamento predefinito di '**`p2c`**' può essere configurato attraverso una serie di file di configurazione:

1. '`/usr/lib/p2c/p2crc`', '**`$P2CRC`**'
2. '`~/p2crc`'
3. '`~/.p2crc`'

Il primo file dell'elenco è quello usato per definire la configurazione generale. Eventualmente, si può usare la variabile di ambiente '**`P2CRC`**', contenente il percorso assoluto per raggiungere un file analogo, sostituendosi in tal modo a quello generale.

Dopo il file di configurazione generale, viene cercato il file '`p2crc`' nella directory personale dell'utente, oppure, in sua mancanza, il file '`.p2crc`'. Questo file serve a definire una personalizzazione della configurazione di '**`p2c`**'.

Le direttive di questo file di configurazione sono rappresentate da assegnamenti, espressi in una delle due forme seguenti.

nome `=` ***valore***

nome ***valore***

I commenti si rappresentano come di consueto facendoli precedere dal simbolo '`#`', dove le righe vuote o bianche vengono semplicemente ignorate.

Il file di configurazione che accompagna Pascal-to-C, cioè '`/usr/lib/p2c/p2crc`', contiene l'elenco completo di tutte le direttive utilizzabili, tutte impostate nel modo più conveniente per l'uso normale e tutte debitamente commentate in modo da sapere come può essere modificato ogni valore.

L'esempio seguente definisce l'utilizzo di un sorgente TURBO Pascal:

```
Language Turbo
```

Le direttive di configurazione possono anche essere incorporate all'interno dello stesso sorgente Pascal, permettendo così una definizione dinamica, riferita a porzioni di codice. Per farlo, si utilizza una forma speciale dei commenti Pascal (le parentesi graffe fanno parte della direttiva).

{***nome=valore***}

In tal caso, come si può vedere, il simbolo '`=`' è obbligatorio e l'uso di spazi bianchi è generalmente inammissibile. È possibile l'utilizzo di commenti anche all'interno di direttive espresse in questo modo. Per farlo, occorre usare la sequenza '`##`'.

La configurazione dinamica all'interno del sorgente, permette di utilizzare anche altre modalità di assegnamento e di eliminazione automatica delle definizioni alla fine del sorgente. Per approfondirle, conviene consultare la documentazione originale, cosa che si riduce in pratica alla lettura di *p2c(1)*.

Segue la descrizione di alcuni esempi.

- `{Language=Turbo}`
  Definisce l'utilizzo di un sorgente TURBO Pascal.
- `{Language=Turbo ## utilizza una codifica TURBO Pascal}`
  Definisce l'utilizzo di un sorgente TURBO Pascal e vi aggiunge un commento interno.

Le direttive della configurazione di Pascal-to-C sono numerose; anche se l'impostazione predefinita si adatta alle situazioni più comuni, potrebbe essere conveniente modificarne alcune, già le prime volte che si utilizza Pascal-to-C.

`AnsiC` [`0`|`1`]

Permette di definire il tipo di dialetto C da utilizzare. Se si attiva la modalità, utilizzando il valore uno, si fa in modo di generare codice C ANSI; se invece non si inserisce, o si utilizza il valore zero, si ottiene un codice compatibile con il C K&R originale.

Come accennato, se non si definisce diversamente, si ottiene un codice C tradizionale, mentre potrebbe essere desiderabile di generare codice C ANSI.

`Language` [`HP`|`HP-UX`|`Turbo`|`UCSD`|`VAX`|`Oregon`|`Berk`|`Modula`]

Permette di definire il dialetto Pascal utilizzato come sorgente per la conversione. Le varie parole chiave usate per distinguere i dialetti hanno il valore seguente:

'**`HP`**'	È la codifica usata in modo predefinito e si riferisce precisamente al Pascal HP, compatibile con il Pascal dello standard ISO.
'**`HP-UX`**'	È il Pascal HP del sistema HP-UX, praticamente identico al Pascal HP normale.
'**`Turbo`**'	TURBO Pascal 5.0, quello usato con il Dos. La differenza rispetto al tipo HP è minima, tanto che generalmente non è necessario richiedere esplicitamente questo tipo di codifica, quando si usano sorgenti TURBO Pascal.
'**`UCSD`**'	UCSD Pascal. Si tratta di un dialetto molto simile al TURBO Pascal.
'**`MPW`**'	Macintosh Programmer's Workshop Pascal 2.0, senza le estensioni Object Pascal.
'**`VAX`**'	VAX/VMS Pascal 3.5. Non tutte le funzionalità sono disponibili.
'**`Oregon`**'	Oregon Software Pascal/2.
'**`Berk`**'	Berkeley Pascal con le estensioni Sun.
'**`Modula`**'	Modula-2. Basato sul libro *Programming in Modula-2* di Wirth, terza edizione. La conversione in C a partire da questo formato non è ancora completa.

`ShortOpt` [`0`|`1`]

Permette di definire il modo con cui devono essere valutate le espressioni logiche: uno abilita il «cortocircuito» attraverso cui si valutano effettivamente solo le condizioni strettamente necessarie a determinare il risultato finale; zero lo disabilita, in modo che tutte le condizioni vengano valutate in ogni caso.

Uso di Pascal-to-C

La conversione del sorgente Pascal in linguaggio C avviene per mezzo del programma '**`p2c`**', configurato come descritto nelle sezioni precedenti.

'**`p2c`**' è effettivamente un compilatore, il cui risultato è un programma C. Questo significa che genera da solo la segnalazione di errori di sintassi nel sorgente Pascal e, alla fine, il sorgente C che si ottiene dovrebbe essere corretto (dal punto di vista del C).

`p2c` [***opzioni***] [***file***]

'**`p2c`**' legge il file indicato come argomento, oppure lo standard input in sua mancanza. In base alle opzioni e alla configurazione definita, genera da quel file una trasformazione in linguaggio C.

Il nome del file generato si ottiene togliendo l'eventuale estensione precedente e aggiungendo '`.c`'.

Tabella u118.5. Alcune opzioni di '**`p2c`**'.

Opzione	Descrizione
`-o` ***file***	Definisce esplicitamente il nome del file del sorgente C da generare.
`-c` ***file_di_configurazione***	Definisce il nome di un file di configurazione da utilizzare al posto di quelli standard.
`-a`	Genera codice C ANSI. Questa opzione permette di sostituirsi agevolmente alla configurazione standard secondo cui il sorgente generato dovrebbe essere di tipo tradizionale (K&R).
`-l {HP|HP-UX|Turbo|UCSD|VAX|Oregon|Berk|Modula}`	Permette di definire il tipo di Pascal nel sorgente. Le caratteristiche abbinate alle varie parole chiave sono state descritte in occasione della descrizione dei file di configurazione.

Segue la descrizione di alcuni esempi.

- `$ p2c mio_programma.pas` [*Invio*]
  Genera il file '`mio_programma.c`' convertendo il contenuto di

‘mio_programma.pas’.

- `$ p2c -a mio_programma.pas` [*Invio*]

  Come nell’esempio precedente, ma genere un programma C secondo lo standard ANSI.

- `$ p2c -a -o mio.c mio_programma.pas` [*Invio*]

  Come nell’esempio precedente, ma il file generato è ‘mio.c’.

Segue la descrizione di un esempio di compilazione:

```
{
    CiaoMondo.pas
    Programma elementare di visualizzazione di un messaggio
    attraverso lo standard output.
}

program CiaoMondo;

begin
    Writeln('Ciao Mondo!');
end.
```

Se il file si chiama ‘CiaoMondo.pas’, si può trasformare in C con il comando seguente:

```
$ p2c CiaoMondo.pas [Invio]

  CiaoMondo

  Translation completed
```

Si ottiene così il file ‘CiaoMondo.c’, mostrato di seguito.

```
/* Output from p2c, the Pascal-to-C translator */
/* From input file "CiaoMondo.pas" */


/*
    CiaoMondo.pas
    Programma elementare di visualizzazione di un messaggio
    attraverso lo standard output.
*/


#include <p2c/p2c.h>


main(argc, argv)
int argc;
Char *argv[];
{
  PASCAL_MAIN(argc, argv);
  printf("Ciao Mondo!\n");
  exit(EXIT_SUCCESS);
}



/* End. */
```

Questo file può essere compilato a sua volta.

```
$ cc -lp2c -o CiaoMondo CiaoMondo.c [Invio]
```

Se tutto funziona correttamente, si ottiene il file ‘CiaoMondo’ eseguibile.

```
$ ./CiaoMondo [Invio]
```

```
Ciao Mondo!
```

Se si desidera generare un sorgente C ANSI, si può usare l’opzione ‘**-a**’ di ‘**p2c**’. Nel caso dell’esempio, il corpo del programma C sarebbe stato il seguente:

```
main(int argc, Char *argv[])
{
  PASCAL_MAIN(argc, argv);
  printf("Ciao Mondo!\n");
  exit(EXIT_SUCCESS);
}
```

GPC

GPC, [2] ovvero il Pascal GNU, è un compilatore Pascal che fa parte di GCC. Il suo utilizzo immediato è molto semplice:

```
gpc file_pascal
```

In questo modo, l’eseguibile ‘**gpc**’ compila il sorgente indicato come argomento e genera il file ‘a.out’, che poi può essere avviato. In alternativa, per specificare il nome del file eseguibile da generare con la compilazione, si usa l’opzione ‘**-o**’:

```
gpc -o file_eseguibile file_pascal
```

Per esempio, per compilare il programma seguente, contenuto nel file ‘CiaoMondo.pas’, si può procedere con il comando mostrato subito dopo:

```
{
    CiaoMondo.pas
    Programma elementare di visualizzazione di un messaggio
    attraverso lo standard output.
}

program CiaoMondo;

begin
    Writeln('Ciao Mondo!');
end.
```

```
$ gpc -o CiaoMondo CiaoMondo.pas [Invio]
```

Dalla compilazione si genera il file eseguibile ‘CiaoMondo’:

```
$ ./CiaoMondo [Invio]
```

```
Ciao Mondo!
```

GPC è un compilatore Pascal molto sofisticato, accompagnato da una documentazione molto dettagliata. Qui si elencano soltanto le opzioni di uso più comune, per consentirne l’uso a scopo didattico, con gli esempi che vengono descritti nei capitoli successivi di questa parte dedicata al Pascal.

Tabella u118.13. Alcune opzioni per l’uso del compilatore GPC.

Opzione	Descrizione
`-o` ***file***	Specifica il nome del file eseguibile o del file oggetto da generare.
`-Wall`	Richiede di mostrare tutte le informazioni diagnostiche (*warning*) che possono essere utili a migliorare il sorgente Pascal.
`-Werror`	Fa in modo che ogni piccolo difetto del sorgente sia considerato alla stregua di un errore che impedisce il buon esito della compilazione.
`-static`	Se possibile, fa in modo di incorporare le librerie nell’eseguibile che viene generato.
`--short-cirquit` `--no-short-cirquit`	Abilita o disabilita il «cortocircuito», ovvero la valutazione effettiva delle sole condizioni strettamente necessarie a determinare il risultato finale.

Free-Pascal

Free-Pascal [3] è un altro compilatore Pascal. La compilazione è svolta dall’eseguibile ‘**fpc**’ che si avvale anche di un file di configurazione, ‘/etc/fpc.cfg’, che può contenere le stesse opzioni della riga di comando.

In condizioni normali, quando si installa Free-Pascal da un pacchetto già pronto per la propria distribuzione GNU, il file di configurazione dovrebbe essere già adatto alle caratteristiche del proprio sistema, senza richiedere altri interventi. Così, di solito è sufficiente compilare un programma in questo modo:

```
fpc file_pascal
```

Di solito, se il nome del file sorgente Pascal ha un’estensione del tipo ‘.pas’, si ottiene un file eseguibile con la stessa radice e senza estensione. Per esempio, compilando il file seguente, denominato ‘CiaoMondo.pas’, si ottiene il file eseguibile ‘ciaomondo’ nella stessa directory:

```
{
    CiaoMondo.pas
    Programma elementare di visualizzazione di un messaggio
    attraverso lo standard output.
}

program CiaoMondo;

begin
    Writeln('Ciao Mondo!');
end.
```

$ **fpc CiaoMondo.pas** [*Invio*]

Si osservi che il nome dell'eseguibile che si ottiene contiene solo lettere minuscole:

$ **./ciaomondo** [*Invio*]

```
Ciao Mondo!
```

[1] **Pascal-to-C** GNU GPL

[2] **GPC** GNU GPL

[3] **Free-Pascal** GNU LGPL

Pascal: introduzione

Struttura fondamentale 959
- Istruzioni Pascal 960
- Nomi 960
- Commenti 960
- Suddivisione di un programma Pascal 960
- Output elementare 961

Variabili e tipi 961
- Valori contenibili e costanti letterali 961
- Dichiarazione delle variabili 962

Operatori ed espressioni 962
- Operatori aritmetici 962
- Operatori di confronto e operatori logici 963

Strutture di controllo del flusso 963
- Struttura condizionale: «if» 964
- Struttura di selezione: «case» 965
- Iterazione con condizione di uscita iniziale: «while» 966
- Iterazione con condizione di uscita finale: «repeat-until» 966
- Iterazione enumerativa: «for» 967

Procedure e funzioni 967
- Struttura 967
- Campo di azione 968
- Forward 968
- Parametri formali e chiamata per valore o per riferimento 969
- Chiamata e parametri attuali 969

I/O elementare 970
- Procedure «Write()» e «Writeln()» 970
- Procedure «Read()» e «Readln()» 971

Struttura del sorgente: le dichiarazioni 971

Riferimenti 972

Il linguaggio Pascal è nato come strumento puramente didattico, che poi si è esteso fino a raggiungere potenzialità vicine a quelle del linguaggio C.

La caratteristica più appariscente di questo linguaggio è che tutto deve essere dichiarato prima del suo utilizzo. Il vantaggio di questo tipo di approccio sta nella possibilità di escludere errori di programmazione dovuti a digitazione errata dei nomi delle variabili, perché il compilatore si rifiuta di considerarle se non sono state dichiarate preventivamente.

Dal momento che di dialetti Pascal ne esistono molti, in questo capitolo si cerca di fare riferimento allo standard ANSI, anche se potrebbe essere particolarmente riduttivo. Gli esempi che vengono proposti dovrebbero essere compatibili con i compilatori descritti nel capitolo u118, senza bisogno di configurazioni particolari.

Struttura fondamentale

Il Pascal impone una struttura nella preparazione dei sorgenti. L'esempio seguente è un programma che non fa alcunché.

```
program Nulla;

begin
end.
```

Nella prima riga dell'esempio, si può osservare la definizione del nome del programma, attraverso la direttiva '**program**'. Il nome, in questo caso è '**Nulla**', non deve corrispondere necessariamente al nome del file.

Le parole chiave ‘**begin**’ e ‘**end**’ delimitano lo spazio utilizzato per le istruzioni del programma, che in questo caso non esistono.

Il punto finale, dopo la parola chiave ‘**end**’, serve a indicare al compilatore la conclusione del programma, che può apparire solo alla fine del sorgente.

Istruzioni Pascal

Le istruzioni Pascal terminano con un punto e virgola (‘**;**’), così un’istruzione può impiegare più righe senza bisogno di utilizzare simboli di continuazione, oppure, su una riga possono apparire più istruzioni (sempre separate con il punto e virgola).

È possibile raggruppare più istruzioni attraverso i delimitatori ‘**begin**’ e ‘**end**’: il primo dei due viene seguito dalle istruzioni senza l’uso del punto e virgola, mentre il secondo termina normalmente con un punto e virgola, oppure un punto se si tratta del delimitatore che conclude il programma.

```
istruzione;
```

```
begin istruzione; istruzione; istruzione; end;
```

L’istruzione nulla può essere rappresentata da un punto e virgola isolato.

Nomi

Secondo il Pascal standard, i nomi che servono per identificare ciò che si utilizza, come variabili, procedure o funzioni, sono composti da una lettera alfabetica, seguita da una combinazione libera di altre lettere e cifre numeriche. Secondo lo standard originale non è ammissibile l’uso del trattino basso, ma la maggior parte dei compilatori ammette anche questo carattere.

La lunghezza dei nomi dovrebbe essere libera, con la limitazione che ogni compilatore è in grado di distinguere i nomi solo in base a un numero massimo di caratteri. Il valore minimo definito dallo standard è di otto caratteri.

Per quanto riguarda i nomi, il Pascal non distingue tra maiuscole e minuscole, come invece avviene nel linguaggio C.

Commenti

Il Pascal consente l’utilizzo di due tipi di delimitatore per circoscrivere i commenti: le parentesi graffe (‘**{**’ e ‘**}**’) e la coppia ‘**(***’ ‘***)**’. Generalmente non sono ammissibili i commenti annidati, cioè quelli a più livelli.

Quello che segue è l’esempio del programma che non fa alcunché, con qualche commento.

```
{
    Ecco un programma che non fa proprio nulla.
}
program Nulla;

begin
    (* è qui che ha luogo il «nulla» *)
end.
```

Esistono due tipi di delimitatori per i commenti solo perché i primi, cioè le parentesi graffe, sono difficili da ottenere nelle prime tastiere di alcuni paesi europei.

Suddivisione di un programma Pascal

Il linguaggio Pascal è un po’ rigido per ciò che riguarda la sequenza con cui possono essere descritte le varie parti che lo compongono. Si distinguono tre parti fondamentali nel file sorgente:

1. intestazione del programma -- si tratta della dichiarazione ‘**program**’ seguita dal nome;
2. dichiarazioni -- è lo spazio in cui si dichiara tutto ciò che viene usato nel programma, per esempio le variabili, le procedure e le funzioni;

3. istruzioni -- è lo spazio, delimitato dalle parole chiave ‘**begin**’ ‘**end**’, in cui si inseriscono le istruzioni del programma, ovvero è quello che in altri linguaggi di programmazione è la funzione o la procedura principale.

È il caso di osservare che i commenti possono essere collocati in ogni punto del file sorgente.

Output elementare

Quasi tutti gli esempi di programmazione elementare, in qualunque linguaggio di programmazione, utilizzano un’istruzione per l’output elementare.

Negli esempi che vengono mostrati inizialmente, si fa spesso uso della procedura ‘**Writeln()**’, la quale si occupa semplicemente di emettere attraverso lo standard output tutti gli argomenti forniti. L’esempio seguente serve a emettere la frase «1000 volte ciao mondo!», utilizzando due parametri: la costante numerica 1000 e la stringa « volte ciao mondo!».

```
program CiaoMondo1000;

begin
    Writeln(1000, ' volte ciao mondo!');
end.
```

Si tenga presente, in ogni caso, che ‘**Writeln**’ e ‘**writeln**’ sono la stessa cosa.

Variabili e tipi

I tipi di dati elementari del linguaggio Pascal dipendono dal compilatore utilizzato e dall’architettura dell’elaboratore sottostante. I tipi standard del Pascal ANSI sono elencati nella tabella u119.4. Il tipo ‘**char**’, non fa parte dello standard ANSI, ma è molto diffuso e così appare incluso in quella tabella.

Tabella u119.4. Elenco dei tipi di dati primitivi fondamentali in Pascal.

Tipo	Descrizione
`int`	Numeri interi positivi e negativi.
`byte`	Interi positivi di un solo byte (da 0 a 255).
`real`	Numeri a virgola mobile.
`boolean`	Valori logici booleani.
`char`	Carattere (generalmente di 8 bit).

Valori contenibili e costanti letterali

Ogni tipo di variabile può contenere un solo tipo di dati, esprimibile eventualmente attraverso una costante letterale scritta secondo una forma adatta.

I valori numerici vengono espressi da costanti letterali senza simboli di delimitazione.

- Gli interi (‘**integer**’) vanno espressi con numeri normali, senza punti di separazione di un’ipotetica parte decimale, prefissati eventualmente dal segno meno (‘**-**’) nel caso di valori negativi.
- I valori ‘**byte**’ vanno espressi come gli interi positivi, con la limitazione della dimensione massima.
- I numeri reali (‘**real**’) possono essere espressi come numeri aventi una parte decimale, segnalata dalla presenza di un punto decimale.

  Se si vuole indicare un numero reale corrispondente a un numero intero, si deve aggiungere un decimale finto, per esempio, il numero 10 si può rappresentare come ‘**10.0**’.

  Naturalmente è ammissibile anche la notazione esponenziale, come per esempio ‘**7e-2**’ che corrisponde in pratica a $7*(10^{-2})$, pari a 0,07 (scritto ‘**0.07**’).

I valori logici vengono espressi dalle costanti letterali '**TRUE**' e '**FALSE**'.

I valori carattere e stringa, vengono delimitati da coppie di apici singoli, come '**'A'**', '**'B'**', ... '**'Ciao Mondo!'**'.

Dichiarazione delle variabili

La dichiarazione delle variabili può essere fatta esclusivamente prima di un blocco '**begin**' '**end**' di un programma, di una funzione o di una procedura.

```
var nome : tipo;
```

Dalla sintassi si vede l'utilizzo della parola chiave '**var**', seguita dal nome della variabile da definire, quindi da due punti ('**:**'), infine dalla definizione del tipo di variabile.

In realtà, è possibile anche indicare un elenco di nomi, separati da virgole, quando questi devono essere tutti dello stesso tipo; inoltre, è possibile dichiarare più variabili differenti, utilizzando la parola chiave '**var**' una sola volta.

Segue la descrizione di alcuni esempi.

- ```
  var     conta   :       integer;
  ```
  Dichiara la variabile '**conta**' di tipo intero.
- ```
  var     conta,canta     :       integer;
  ```
  Dichiara le variabili '**conta**' e '**canta**' di tipo intero.
- ```
  var     conta   :       integer;
          canta   :       integer;
  ```
  Esattamente uguale all'esempio precedente.
- ```
  var
          conta   :       integer;
          lettera :       char;
  ```
  Dichiara la variabile '**conta**' di tipo intero e la variabile '**lettera**' di tipo carattere.

Operatori ed espressioni

Gli operatori sono qualcosa che esegue un qualche tipo di funzione, su uno o due operandi, restituendo un valore. Il tipo di valore restituito varia a seconda dell'operatore e degli operandi utilizzati. Per esempio, la somma di due interi genera un intero, mentre una divisione di un valore intero per un altro numero intero, genera un numero reale.

Operatori aritmetici

Gli operatori che intervengono su valori numerici sono elencati nella tabella u119.9.

Tabella u119.9. Elenco degli operatori aritmetici e di quelli di assegnamento relativi a valori numerici.

Operatore e operandi	Descrizione
op1 `+` ***op2***	Somma i due operandi.
op1 `-` ***op2***	Sottrae dal primo il secondo operando.
op1 `*` ***op2***	Moltiplica i due operandi.
op1 `/` ***op2***	Divide il primo operando per il secondo, il risultato è in virgola mobile.
op1 `div` ***op2***	Divide il primo operando per il secondo generando un risultato intero.
op1 `mod` ***op2***	Modulo: il resto della divisione tra il primo e il secondo operando.
var `:=` ***valore***	Assegna alla variabile il valore alla destra.

Una caratteristica fondamentale del Pascal è la sua attenzione nella coerenza dei tipi di dati utilizzati nelle espressioni e negli assegnamenti. Tanto per comprendere il problema con un esempio, un compilatore non dovrebbe consentire l'assegnamento di un valore in virgola mobile in una variabile intera. Naturalmente, ogni compilatore può utilizzare una politica differente, consentendo una conversione di tipo automatica in situazioni particolari.

In ogni caso, è necessario conoscere l'uso di alcune funzioni essenziali, utili per prevenire conflitti nel tipo dei dati.

```
Round(numero_reale)
```

```
Trunc(numero_reale)
```

Le due funzioni, usate in questo modo, restituiscono un valore intero a partire da un valore a virgola mobile. Nel primo caso il numero viene arrotondato, mentre nel secondo viene semplicemente troncato al valore intero.

Operatori di confronto e operatori logici

Gli operatori di confronto determinano la relazione tra due operandi. Il risultato dell'espressione composta da due operandi messi a confronto è di tipo booleano, rappresentabile in Pascal con le costanti '**TRUE**' e '**FALSE**'. Gli operatori di confronto sono elencati nella tabella u119.10.

Tabella u119.10. Elenco degli operatori di confronto. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 `=` ***op2***	*Vero* se gli operandi si equivalgono.
op1 `!=` ***op2***	*Vero* se gli operandi sono differenti.
op1 `<` ***op2***	*Vero* se il primo operando è minore del secondo.
op1 `>` ***op2***	*Vero* se il primo operando è maggiore del secondo.
op1 `<=` ***op2***	*Vero* se il primo operando è minore o uguale al secondo.
op1 `>=` ***op2***	*Vero* se il primo operando è maggiore o uguale al secondo.

Quando si vogliono combinare assieme diverse espressioni logiche, comprendendo in queste anche delle variabili che contengono un valore booleano, si utilizzano gli operatori logici. Gli operatori logici sono elencati nella tabella u119.11.

Tabella u119.11. Elenco degli operatori logici. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
`not` ***op***	Inverte il risultato logico dell'operando.
op1 `and` ***op2***	*Vero* se entrambi gli operandi restituiscono il valore *Vero*.
op1 `or` ***op2***	*Vero* se uno o entrambi gli operandi restituiscono il valore *Vero*.

Nel Pascal tradizionale, le espressioni logiche vengono valutate in ogni parte, prima di definire il risultato finale di un operatore AND o di un operatore OR. Dal momento che questo metodo di risoluzione è inutilmente dispersivo, spesso i compilatori Pascal consentono di ottenere il «cortocircuito», attraverso cui si valutano solo le parti dell'espressione che sono indispensabili per arrivare al risultato finale.

Strutture di controllo del flusso

Il linguaggio Pascal gestisce un buon numero di strutture di controllo di flusso, compreso il salto *go-to* che comunque è sempre meglio non utilizzare e qui, volutamente, non viene presentato.

Le strutture di controllo permettono di sottoporre l'esecuzione di una parte di codice alla verifica di una condizione, oppure permettono di

eseguire dei cicli, sempre sotto il controllo di una condizione. La parte di codice che viene sottoposta a questo controllo, può essere un'istruzione singola, oppure un gruppo di istruzioni. Nel secondo caso, quasi sempre, è necessario delimitare questo gruppo attraverso l'uso di '**begin**' e '**end**'.

Dal momento che è comunque consentito di realizzare un gruppo di istruzioni che in realtà ne contiene una sola, probabilmente è meglio utilizzare sempre i delimitatori '**begin**' '**end**', a vantaggio dello stile e della leggibilità del codice.

Struttura condizionale: «if»

```
if condizione then istruzione
```

```
if condizione then istruzione else istruzione
```

Se la condizione si verifica, viene eseguita l'istruzione (o il gruppo di istruzioni) seguente; quindi il controllo passa alle istruzioni successive alla struttura. Se viene utilizzato '**else**', nel caso non si verifichi la condizione, viene eseguita l'istruzione che ne segue. Vengono mostrati alcuni esempi.

```
…
var     importo : integer;
…
if importo > 10000000 then Writeln( 'offerta vantaggiosa' );
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
begin
    memorizza := importo;
    Writeln( 'offerta vantaggiosa' );
end
else
    Writeln( 'meglio lasciar perdere' );
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
begin
    memorizza := importo;
    Writeln( 'offerta vantaggiosa' );
end
else if importo > 5000000 then
begin
    memorizza := importo;
    Writeln( 'offerta accettabile' );
end
else
    Writeln( 'meglio lasciar perdere' );
```

Il blocco *if-then-else* rappresenta un'unica istruzione in Pascal. In questo senso, dovrebbe apparire un punto e virgola alla fine del blocco, a terminare l'istruzione. Se si utilizzano raggruppamenti di istruzioni attraverso i delimitatori '**begin**' '**end**', le istruzioni contenute terminano con il punto e virgola, mentre il blocco, dopo la parola chiave '**end**', no, a meno che si tratti della fine dell'istruzione '**if**'.

Per osservare meglio questo particolare, si potrebbero riscrivere gli stessi esempi nel modo seguente, in cui il punto e virgola finale serve a concludere visivamente la dentellatura delle istruzioni '**if**'.

```
…
var     importo : integer;
…
if importo > 10000000 then
    Writeln( 'offerta vantaggiosa' )
;
```



```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
    begin
        memorizza := importo;
        Writeln( 'offerta vantaggiosa' );
    end
else
    Writeln( 'meglio lasciar perdere' )
;
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
    begin
        memorizza := importo;
        Writeln( 'offerta vantaggiosa' );
    end
else
    if importo > 5000000 then
        begin
            memorizza := importo;
            Writeln( 'offerta accettabile' );
        end
    else
        Writeln( 'meglio lasciar perdere' )
;
```

Struttura di selezione: «case»

La struttura di selezione si ottiene con l'istruzione '**case**'. Si tratta di una struttura un po' troppo complessa per essere rappresentata facilmente attraverso uno schema sintattico. In generale, l'istruzione '**case**' permette di eseguire una o più istruzioni in base al risultato di un'espressione. L'esempio seguente mostra la visualizzazione del nome del mese, in base al valore di un intero.

```
…
var     mese    : integer;
…
case mese of
    1  : Writeln( 'gennaio' );
    2  : Writeln( 'febbraio' );
    3  : Writeln( 'marzo' );
    4  : Writeln( 'aprile' );
    5  : Writeln( 'maggio' );
    6  : Writeln( 'giugno' );
    7  : Writeln( 'luglio' );
    8  : Writeln( 'agosto' );
    9  : Writeln( 'settembre' );
    10 : Writeln( 'ottobre' );
    11 : Writeln( 'novembre' );
    12 : Writeln( 'dicembre' );
end;
```

È importante osservare l'uso del punto e virgola, che conclude ogni istruzione richiamata dai vari casi. La parola chiave '**end**' finale, conclude la struttura.

Un gruppo di casi può essere raggruppato assieme, quando si vuole che ognuno di questi esegua lo stesso insieme di istruzioni:

```
…
var     anno    : integer;
        mese    : integer;
        giorni  : integer;
…
case mese of
    1,3,5,7,8,10,12 :
        giorni := 31;
    4,6,9,11 :
        giorni := 30;
    2   :
        if ((anno mod 4 = 0) and not (anno mod 100 = 0)) or
                (iAnno mod 400 = 0) then
            giorni := 29
        else
            giorni := 28
        ;
end;
```

È anche possibile definire un caso predefinito che si verifichi quando nessuno degli altri si avvera:

```
…
var     mese    : integer;
…
case mese of
    1  : Writeln( 'gennaio' );
    2  : Writeln( 'febbraio' );
…
    11 : Writeln( 'novembre' );
    12 : Writeln( 'dicembre' );
else
    Writeln( 'mese non corretto' );
end;
```

Un intervallo di casi può essere indicato facilmente come nell'esempio seguente:

```
…
var     mese    : integer;
…
case mese of
    6..9 : Writeln( 'mesi caldi' );
    …
end;
```

Iterazione con condizione di uscita iniziale: «while»

```
while condizione do istruzione
```

La struttura '**`while`**' esegue un'istruzione finché la condizione restituisce il valore *Vero*. La condizione viene valutata prima di eseguire l'istruzione e poi ogni volta che termina un ciclo, prima dell'esecuzione del successivo.

Come sempre, al posto della singola istruzione se ne può inserire un raggruppamento, delimitato dalle parole chiave '**`begin`**' e '**`end`**'. L'esempio seguente fa apparire per 10 volte la lettera «x»:

```
program DieciX;

var contatore   : integer;

begin
    contatore := 0;
    while contatore < 10 do
    begin
        contatore := contatore + 1;
        Writeln( 'x' );
    end;
end.
```

La struttura '**`while`**' è un'istruzione singola in Pascal. Per sottolinearlo, si potrebbe cambiare la dentellatura dell'esempio appena mostrato per fare in modo che il punto e virgola finale, che chiude l'istruzione, inizi sulla stessa colonna della parola chiave '**`while`**'.

```
…
    contatore := 0;
    while contatore < 10 do
        begin
            contatore := contatore + 1;
            Writeln( 'x' );
        end
    ;
…
```

Iterazione con condizione di uscita finale: «repeat-until»

```
repeat istruzione;… until condizione;
```

La struttura '**`repeat`**' '**`until`**' permette di eseguire un gruppo di istruzioni una volta e poi di ripeterne l'esecuzione fino a quando la condizione posta alla fine continua a non verificarsi.

Ci sono quindi due diversità fondamentali, rispetto alla struttura '**`while`**': il gruppo di istruzioni viene eseguito sicuramente almeno una volta; il verificarsi della condizione implica l'interruzione del ciclo.

Per quanto riguarda la sintassi usata dal Pascal, c'è da osservare che dopo la parola chiave '**`repeat`**' possono essere collocate una serie di istruzioni, senza bisogno di un raggruppamento '**`begin`**' '**`end`**'. In questo senso, ogni istruzione termina con il suo punto e virgola.

L'esempio seguente è solo un pretesto per mostrare il funzionamento di questa struttura: visualizza 10 volte la lettera «x».


```
program DieciX;

var contatore   : integer;

begin
    contatore := 0;
    repeat
        contatore := contatore + 1;
        Writeln( 'x' );
    until contatore = 10;
end.
```

Iterazione enumerativa: «for»

```
for variabile; := inizio to fine do istruzione
```

L'istruzione '**`for`**' permette di definire un ciclo enumerativo, in cui una variabile intera viene inizializzata, quindi viene eseguita ripetitivamente l'istruzione controllata, incrementando alla fine di ogni esecuzione tale variabile e interrompendo il ciclo quando questa raggiunge il valore finale (quando la variabile ha raggiunto il valore finale, si esegue l'istruzione per l'ultima volta). L'incremento è di un'unità quando il valore finale è maggiore di quello iniziale, oppure di un'unità negativa quando il valore finale è minore di quello iniziale.

L'esempio già visto, in cui veniva visualizzata per 10 volte una «x», potrebbe tradursi nel modo seguente, attraverso l'uso di un ciclo '**`for`**'.

```
program DieciX;

var contatore   : integer;

begin
    for contatore := 1 to 10 do
        Writeln( 'x' )
    ;
end.
```

Come sempre, al posto di controllare una singola istruzione, se ne può gestire un gruppo, attraverso l'uso dei delimitatori '**`begin`**' e '**`end`**'. L'esempio già visto, potrebbe eventualmente tradursi nel modo seguente:

```
…
    for contatore := 1 to 10 do
        begin
            Writeln( 'x' );
        end
    ;
…
```

Procedure e funzioni

Il linguaggio Pascal distingue due tipi di subroutine: procedure e funzioni. In pratica, le procedure sono funzioni che non restituiscono alcun valore.

La dichiarazione e descrizione delle procedure e delle funzioni deve essere fatta all'interno della parte iniziale del programma, dedicata alle dichiarazioni. Procedure e funzioni possono chiamarsi a vicenda e, in ogni caso, perché la chiamata possa essere valida, occorre che la procedura o la funzione sia stata dichiarata precedentemente.

Ci sono situazioni in cui non è possibile descrivere una funzione o una procedura prima di quella chiamante. In tali casi, è possibile dichiarare una funzione senza descriverla immediatamente.

Struttura

Per il linguaggio Pascal, le procedure e le funzioni sono dei sottoprogrammi veri e propri, tanto che anche in questo caso si distinguono tre parti: intestazione, dichiarazioni e istruzioni. In particolare, l'intestazione può includere anche la dichiarazione, a meno che questa non sia separata per renderla visibile ad altre procedure e funzioni precedenti.

```
procedure nome[(parametro_formale[…])];
```

```
function nome[(parametro_formale[…])] : tipo;
```

La sintassi che appare sopra rappresenta la dichiarazione di una procedura e di una funzione. Come si può osservare, a parte la parola chiave iniziale, la funzione ha alla fine l'indicazione del tipo di dati che restituisce.

Se la procedura o la funzione non richiede l'indicazione di parametri, allora non è necessario specificare alcun ***parametro formale***, quindi non sono necessarie nemmeno le parentesi tonde.

Dopo la dichiarazione della funzione o della procedura, vanno indicate le dichiarazioni, per esempio le variabili utilizzate, nello stesso modo già visto per il programma.

Infine vanno poste le istruzioni, all'interno di un raggruppamento '**begin**' '**end**'. A differenza del raggruppamento analogo che riguarda il blocco principale del programma, la parola chiave '**end**' è conclusa con un punto e virgola invece che con il punto.

La funzione restituisce un valore, attraverso l'assegnamento a una variabile ipotetica che ha lo stesso nome della funzione.

Segue la descrizione di alcuni esempi.

- ```
  procedure CiaoCiao;
  begin
      Writeln('Ciao a tutti');
      Writeln('ciao ciao ciao');
  end;
  ```
  Si tratta di una procedura elementare che non utilizza alcun parametro e si limita a emettere un messaggio di saluto.
- ```
  function CiaoCiao : boolean;
  begin
      Writeln('Ciao a tutti');
      Writeln('ciao ciao ciao');
      CiaoCiao := TRUE;
  end;
  ```
  Si tratta di una funzione elementare che non utilizza alcun parametro e si limita a emettere un messaggio di saluto, restituendo sempre il valore booleano *Vero*.

Campo di azione

Sia le variabili che le procedure e le funzioni, hanno un campo di azione. Le variabili dichiarate nella parte introduttiva di un programma, prima della dichiarazione di procedure e funzioni, sono accessibili al corpo del programma e a tutte le procedure e funzioni. Le variabili dichiarate nella parte introduttiva di una procedura o di una funzione, hanno effetto locale, non essendo visibili all'esterno; se queste hanno nomi già utilizzati per le variabili globali, di fatto ne impediscono l'accesso.

Le procedure e le funzioni, in qualità di sottoprogrammi, possono contenere anche la dichiarazione di sottoprocedure e sottofunzioni. In tal caso, tali subroutine sono accessibili solo dal codice contenuto nella procedura o funzione in cui sono dichiarate. Nello stesso modo, le variabili locali delle procedure o delle funzioni sono accessibili anche alle rispettive sottoprocedure e sottofunzioni.

Forward

Si è accennato al fatto che, perché una chiamata possa essere valida, occorre che la procedura o la funzione in questione sia stata dichiarata prima, cioè in una posizione precedente all'interno del sorgente.

In presenza di chiamate ricorsive tra più procedure o funzioni, diviene impossibile che ogni chiamata si riferisca sempre a qualcosa di definito e descritto in precedenza.

Per risolvere il problema, si può dichiarare una procedura o una funzione prima della sua descrizione effettiva, attraverso l'uso della parola chiave '**forward**', come nell'esempio seguente:

```
…
procedure MiaProcedura(…);
forward;
…
…
procedure MiaProcedura;
begin
    …
end;
…
```


La dichiarazione della procedura o della funzione deve contenere la dichiarazione di tutti i parametri formali, mentre la descrizione è assente.

Parametri formali e chiamata per valore o per riferimento

La descrizione dei parametri formali, all'interno della dichiarazione di una procedura o di una funzione, richiede la definizione del nome delle variabili e del tipo relativo. Il campo di azione di queste variabili è locale.

```
…
procedure MiaProcedura( primo,secondo : integer;
                        terzo         : char);
begin
    …
end;
…
```

L'esempio mostra la dichiarazione di una procedura che utilizza tre parametri formali, denominati casualmente proprio: '**primo**', '**secondo**' e '**terzo**'. I primi due sono di tipo '**integer**', mentre l'ultimo è di tipo '**char**'.

Come si può osservare, la dichiarazione dei parametri formali è molto simile alla dichiarazione delle variabili, con la differenza che ciò avviene all'interno di parentesi tonde, oltre al fatto che (per il momento) manca la parola chiave '**var**'.

Una procedura o una funzione in cui i parametri formali siano stati dichiarati in questo modo, riceve una copia dei dati nel momento della chiamata, senza poter riflettere all'indietro le modifiche che a questi dovesse applicare. Si ha in pratica una chiamata per valore.

È possibile dichiarare una procedura o una funzione in cui la chiamata sia per riferimento, in modo da riflettere all'indietro le modifiche, utilizzando la parola chiave '**var**'.

```
…
procedure MiaProcedura( primo       : integer;
                        var secondo : integer;
                        terzo       : char);
begin
    …
end;
…
```

L'esempio mostra una variante in cui si dichiara che il secondo parametro formale, '**secondo**', riflette all'indietro le modifiche che dovessero essergli apportate all'interno della procedura.

Chiamata e parametri attuali

La chiamata di una procedura o di una funzione, avviene semplicemente nominandola e facendola seguire dall'indicazione dei ***parametri attuali***, cioè dei valori che si vuole siano passati per l'elaborazione.

La differenza fondamentale tra procedure e funzioni sta nel fatto che le chiamate alle prime vengono utilizzate come istruzioni pure e semplici, mentre le seconde, vanno inserite all'interno di espressioni.

Merita un minimo di attenzione anche il tipo di chiamata: per valore o per riferimento. Nel primo caso, non si pongono problemi di alcun tipo, dal momento che la funzione o la procedura chiamata non può alterarli; se invece si tratta di una chiamata per riferimento, occorre fare attenzione che il parametro attuale, usato nella chiamata, non sia una costante, perché questo genererebbe un errore irreversibile.

```
…
var     MioNumero : integer;
…
procedure MiaProcedura( primo       : integer;
                        var secondo : integer;
                        terzo       : char);
begin
    …
    secondo := 777;
    …
end;
…
{ inizio del programma }
begin
    MiaProcedura( 123, MioNumero, 'C' );
    Writeln( MioNumero );
end.
```

L'esempio mostra una chiamata a una procedura in cui uno dei parametri deve essere chiamato per riferimento. In tal caso, il parametro attuale corrispondente, utilizzato nella chiamata, è necessariamente una variabile.

I/O elementare

Per le operazioni di I/O elementare, cioè per l'utilizzo di standard output e standard error, si hanno a disposizione due coppie di procedure: '**`Write()`**' e '**`Writeln()`**'; '**`Read()`**' e '**`Readln()`**'. La prima coppia per emettere qualcosa attraverso lo standard output, la seconda per leggere qualcosa dallo standard input.

Anche se non è ancora stato affrontato l'argomento stringhe, è opportuno anticipare che per inserire un apice singolo all'interno di una costante stringa, basta indicarne due consecutivi. Per esempio, la stringa seguente,

```
'questa è la ''vera'' verità'
```

Si traduce in:

```
questa è la 'vera' verità
```

Procedure «Write()» e «Writeln()»

```
Write(elemento_da_visualizzare [ : dimensione [ : decimali ] ] [ , … ] )
```

```
Writeln(elemento_da_visualizzare [ : dimensione [ : decimali ] ] [ , … ] )
```

Le procedure '**`Write()`**' e '**`Writeln()`**' permettono di emettere attraverso lo standard output il contenuto di tutti i parametri che gli vengono forniti. A seconda dei tipi di dati utilizzati, vengono effettuate tutte le conversioni necessarie a ottenere un risultato stringa.

Se un parametro attuale, fornito nella chiamata, viene indicato seguito da due punti ('`:`') e quindi da un numero, si stabilisce lo spazio (espresso in colonne) che questo deve utilizzare nell'output. Se si specifica tale dimensione, l'informazione viene rappresentata allineandola a destra. Questa possibilità di definire la dimensione viene utilizzata prevalentemente per i dati numerici e in questo senso sta la logica dell'allineamento a destra.

Se si vuole rappresentare un valore numerico con decimali, è abbastanza importante fissare la dimensione della visualizzazione, aggiungendo anche l'indicazione delle colonne da riservare alla parte decimale. Diversamente, la rappresentazione risulterebbe in notazione esponenziale.

L'unica differenza tra le due procedure, sta nel fatto che '**`Writeln()`**' aggiunge automaticamente, alla fine della stringa visualizzata, il codice di interruzione di riga, in modo da riportare il cursore all'inizio della riga successiva.

Segue la descrizione di alcuni esempi.

- ```
  var totale : integer;
  …
  totale := 1950000;
  …
  Write('Totale:', totale:11);
  ```
  Emette la stringa seguente, senza portare a capo il cursore alla fine:

  ```
  Totale:    1950000
  ```

- ```
  var totale : real;
  …
  real := 1234.5678;
  …
  Writeln('Totale:', totale:11:5);
  ```
  Emette la stringa seguente, portando a capo il cursore alla fine.

  ```
  Totale: 1234.56780
  ```


Procedure «Read()» e «Readln()»

```
Read( variabile [ , … ] )
```

```
Readln( variabile [ , … ] )
```

Le procedure '**`Read()`**' e '**`Readln()`**' permettono di leggere dallo standard input dei valori per le variabili che vengono indicate come parametri della chiamata. I dati inseriti, vengono distinti in base all'inserimento di spaziature, così come avviene di solito con gli argomenti di un comando del sistema operativo.

È importante che i dati inseriti siano compatibili con il tipo delle variabili utilizzate, altrimenti si rischia di ottenere un errore irreversibile durante il funzionamento del programma.

La differenza tra le due procedure sta nel fatto che '**`Readln()`**' dovrebbe restituire l'eco del codice di interruzione di riga, quando si preme [*Invio*] per concludere l'inserimento dei dati, mentre '**`Read()`**' no. In pratica, può darsi che il compilatore non riesca a distinguere tra le due procedure, comportandosi sempre nello stesso modo.

Segue la descrizione di alcuni esempi.

- ```
  var totale : integer;
  …
  Write('Inserisci il totale: ');
  Read(totale);
  …
  ```
  Emette l'invito a inserire un valore e quindi lo attende dallo standard input.

- ```
  var capitale : integer;
  var tasso    : real;
  …
  Write('Inserisci di seguito il capitale e il tasso: ');
  Read(capitale,tasso);
  …
  ```
  Emette l'invito a inserire due valori consecutivi: un intero e un valore decimale.

Struttura del sorgente: le dichiarazioni

È già stato accennato alla struttura di un sorgente Pascal: del programma, delle procedure e delle funzioni. Si tratta di tre parti fondamentali:

1. intestazione del programma, dichiarazione della procedura o della funzione;
2. dichiarazioni;
3. istruzioni.

Il punto più delicato è la definizione della parte delle dichiarazioni, dato che nel Pascal originale esiste un ordine preciso nel tipo di istruzioni che possono esservi inserite. Si tratta di dichiarazioni:

1. '**`label`**'
2. '**`const`**'
3. '**`type`**'
4. '**`var`**'
5. '**`procedure`**'
6. '**`function`**'

La maggior parte di queste dichiarazioni non è ancora stata descritta. In particolare, '**`label`**', dal momento che serve a realizzare dei salti incondizionati senza ritorno (*go-to*), non viene descritta in questi capitoli sul Pascal.

Riferimenti

- Gordon Dodrill, *Pascal Language Tutorial*
  http://www.geocities.com/hotdogcom/ptutor/paslist.html
  http://packetstormsecurity.nl/programming-tutorials/Pascal/pascal-tutorial/paslist.htm

Pascal: tipi di dati derivati

Array 973
Dichiarazione e accesso 973
Scansione di un array 974
Stringhe 974
Tipi 975
Costanti 975
Tipo enumerativo, sottointervallo e insieme 976
Tipo enumerativo 976
Sottointervallo 976
Insieme 977
Record 977
With 978
Riferimenti 978

Nel capitolo introduttivo è stato visto l'uso di variabili identificabili semplicemente con il loro nome. La programmazione elementare richiede anche l'utilizzo di strutture di dati più complesse; le stesse stringhe sono degli array di caratteri e come tali vanno trattate.

Array

Gli array in Pascal sono una sequenza ordinata, in una quantità prestabilita, di elementi dello stesso tipo. Gli elementi possono essere composti da qualunque tipo di dati, nativo o derivato.

Una caratteristica importante del linguaggio Pascal sta nel fatto che nel momento della dichiarazione di un array, viene definito anche il valore iniziale dell'indice da utilizzare per la scansione dei vari elementi.

Dichiarazione e accesso

```
var nome : array[inizio..fine] of tipo
```

La sintassi indicata, dove le parentesi quadre fanno parte dell'istruzione, mostra in breve in che modo si possa dichiarare un array, a una sola dimensione, di elementi di un certo tipo di dati.

È importante osservare che vengono stabiliti in modo esplicito sia l'indice iniziale del primo elemento, sia quello finale dell'ultimo, stabilendo implicitamente la quantità di questi elementi.

L'esempio seguente mostra la dichiarazione di tre array simili, composti tutti da sette interi, dove, rispettivamente, il primo elemento si raggiunge con l'indice iniziale 1, 0 e 2.

```
var elenco  : array[1..7] of integer;
    elenco2 : array[0..6] of integer;
    elenco3 : array[2..8] of integer;
```

Per accedere agli elementi di un array si usa la sintassi seguente e anche qui le parentesi quadre fanno parte dell'istruzione.

```
nome[indice]
```

Quello che conta è che l'indice indicato sia valido, in funzione della dichiarazione fatta in origine. L'esempio seguente assegna al primo elemento il valore 10.

```
elenco[1] := 10;
```

Gli array multidimensionali non sono altro che array di array. Il modo più semplice per dichiarare un array multidimensionale è quello di indicare due o più intervalli di valori per gli indici, secondo la sintassi seguente:

```
var nome : array[inizio..fine, inizio..fine...] of tipo
```

Per esempio, l'istruzione seguente dichiara un array a due dimensioni di tre elementi per otto, di tipo intero. Si osservi in particolare il secondo intervallo di indici, dove il primo elemento viene raggiunto con l'indice zero.

```
var elenco : array[1..3,0..7] of integer;
```

In modo analogo, si raggiunge un elemento di un array multidimensionale utilizzando due o più indici, secondo la sintassi seguente:

nome [***indice*** , ***indice***…]

L'esempio seguente assegna un valore all'elemento «1,0».

```
elenco[1,0] := 10;
```

Scansione di un array

La scansione di un array avviene generalmente con un ciclo enumerativo, '**`for`**', come nell'esempio seguente:

```
…
var indice : integer;
var elenco : array[1..7] of integer;
…
begin
    …
    for indice := 1 to 7 do begin
        …
        elenco[indice] := …
        …
    end;
    …
end.
```

La scansione di array multidimensionali avviene generalmente attraverso una serie di cicli enumerativi, uno per ogni dimensione, annidati opportunamente. L'esempio seguente mostra la scansione di un array a tre dimensioni.

```
…
var i,j,k : integer;
var elenco : array[1..7,0..8,2..10] of integer;
…
begin
    …
    for i := 1 to 7 do begin
        …
        for j := 0 to 8 do begin
            …
            for k := 2 to 10 do begin
                …
                elenco[i,j,k] := …
                …
            end;
            …
        end;
        …
    end;
    …
end.
```

Stringhe

Nel linguaggio Pascal, così come in molti altri, le stringhe sono semplicemente degli array di caratteri, con qualche piccola differenza per facilitarne l'utilizzo.

La dichiarazione di una variabile stringa è quindi la dichiarazione di un array composto da una quantità predefinita di caratteri. Nell'esempio seguente, viene creato una variabile stringa di 20 caratteri.

```
var cognome : array[1..20] of char;
```

La variabile dichiarata in questo modo può essere usata come un array, cioè accedendo alle informazioni carattere per carattere, oppure nel suo insieme. Nell'esempio seguente si assegna un nome alla variabile stringa mostrata sopra.

```
cognome := 'Rossi';
```

Se si utilizza un assegnamento di questo tipo, vengono ricoperti anche gli elementi successivi alla lunghezza della stringa letterale assegnata. Quindi, seguendo l'esempio, l'array riceve il nome «Rossi» nei suoi primi cinque elementi, mentre negli altri viene comunque inserito uno spazio.

Tipi

Il linguaggio Pascal permette di definire dei tipi di dati derivati, a partire da quelli elementari, o a partire da altri tipi composti dichiarati precedentemente.

`type` ***tipo_nuovo*** `=` ***definizione_del_tipo***

La definizione di un nuovo tipo va posta nella zona dichiarativa del programma, della procedura o della funzione. L'esempio seguente serve a dichiarare il tipo «Numero» come equivalente al tipo intero standard.

```
type Numero = integer;
```

Naturalmente, la definizione di un nuovo tipo è sensata quando serve a individuare qualcosa di più complesso dei dati elementari, come nel caso di un array. L'esempio seguente dichiara il tipo «Stringa» come un array di 80 caratteri, quindi dichiara il tipo «Nominativo» come array composto da due elementi «Stringa» (probabilmente uno per il nome e l'altro per il cognome).

```
type Stringa = array[1..80] of char;
type Nominativo = array[1..2] of Stringa;
```

A questo punto, per seguire l'esempio, se si generasse una variabile di tipo «Nominativo», si otterrebbe un array di due elementi, che in realtà sono array di 80 caratteri.

```
…
var Nome : Nominativo;
…
begin
    …
    Nome[1] := 'Pinco';
    Nome[2] := 'Pallino';
    …
end.
```

L'esempio mostra in che modo si potrebbe usare una variabile del genere. Tuttavia, si potrebbe accedere anche al singolo elemento carattere, utilizzando due indici.

```
    …
    Nome[1,1] := 'P';
    Nome[1,2] := 'i';
    Nome[1,3] := 'n';
    Nome[1,4] := 'c';
    Nome[1,5] := 'o';
    …
end.
```

Convenzionalmente, quando si dichiara un nuovo tipo di dati, si usa l'iniziale maiuscola, per distinguerlo facilmente dagli altri tipi nativi.

Costanti

Il linguaggio Pascal offre qualcosa di simile alle costanti macro di altri linguaggi come il C. Non si tratta di un linguaggio di precompilazione, ma proprio del Pascal, anche se si tratta comunque di costanti letterali, senza la definizione di un tipo a priori.

`const` ***nome_della_costante*** `=` ***valore_letterale***

La dichiarazione di queste costanti va fatta, come prevedibile, nella zona dichiarativa del programma, della procedura o della funzione. L'esempio seguente dichiara la costante «DIMENSIONE», che poi viene usata per definire la dimensione di una serie di array.

```
…
const DIMENSIONE = 11;
…
var elenco  : array[1..DIMENSIONE] of integer;
    elenco2 : array[1..DIMENSIONE] of integer;
    elenco3 : array[1..DIMENSIONE] of integer;
```

Il vantaggio di utilizzare le costanti sta nel facilitare la lettura del sorgente, nel riconoscere il significato di determinate costanti e nel facilitare la modifica di tali valori, senza dover rileggere tutto il sorgente alla loro ricerca.

Tipo enumerativo, sottointervallo e insieme

Il linguaggio Pascal offre dei tipi di dati particolari, che non sono ancora stati descritti, il cui scopo è solo quello di facilitare il compito del programmatore.

Tipo enumerativo

Il tipo enumerativo, o scalare, secondo la terminologia del Pascal, è una forma di rappresentazione di un intero attraverso costanti mnemoniche. In pratica, si definisce una variabile che può assumere un elenco di valori simbolici possibili, valori che in realtà sono solo delle costanti simboliche, senza un legame con il termine letterale usato per distinguerle, salva la convenienza del programmatore.

(*costante* , *costante* [, …])

La sintassi indicata mostra il modo in cui si definisce un tipo del genere: all'interno di parentesi tonde si elencano i nomi delle costanti che possono essere assegnate a una variabile di questo tipo.

L'esempio seguente mostra la dichiarazione di una variabile scalare che può assumere i valori «VERDE», «BLU» e «ROSSO».

```
var colore : (VERDE, BLU, ROSSO);
```

L'esempio stesso dovrebbe chiarire l'utilità di questo tipo di dati: si lascia al compilatore il compito di stabilire i valori più appropriati per i simboli che possono essere associati a una variabile. Tuttavia, è importante chiarire che non è possibile visualizzare il contenuto di una variabile del genere, in quanto questo non è prevedibile.

```
if colore = VERDE then
    begin
        …
        Writeln( "Il colore è verde" );
    end;
else
    …
;
```

Naturalmente, questo tipo di dati si presta particolarmente per la definizione di tipi derivati, come nell'esempio seguente, dove prima si dichiara un tipo e più avanti si utilizza nella dichiarazione di una nuova variabile.

```
type Sapore = (INSIPIDO, DOLCE, SALATO, ACIDO, PICCANTE, AMARO);
…
var pietanza : Sapore;
…
```

Sottointervallo

Il sottointervallo è la definizione di un tipo derivato che può utilizzare solo un intervallo stabilito di valori. Questo intervallo si definisce solo con l'indicazione di due costanti dello stesso tipo, separate da due punti in sequenza.

Per esempio, per indicare la serie di numeri interi che va da uno a sette, si può utilizzare la notazione '**`1..7`**', mentre per indicare la serie delle lettere alfabetiche minuscole, si può utilizzare la notazione '**`'a'..'z'`**'.

Naturalmente, si possono indicare anche degli intervalli di un tipo enumerativo dichiarato in precedenza. Seguono alcuni esempi.

```
type Settimana = (LUNEDÌ, MARTEDÌ, MERCOLEDÌ,
                  GIOVEDÌ, VENERDÌ, SABATO, DOMENICA);

type Feriale   = LUNEDÌ..VENERDÌ;
…
var lavoro    : Feriale;
    minuscola : 'a'..'z';
…
```

Le variabili dichiarate in questo modo, ottengono dal compilatore il tipo più adatto a contenere l'informazione indicata, senza la necessità di doverlo indicare in modo esplicito.

Insieme

Una variabile può contenere un'informazione riferita a un insieme di elementi enumerativi. In pratica, si tratta di un tipo simile a quello enumerativo, dove ogni elemento può essere presente o meno. Si dichiara questo tipo di dati con le parole chiave '**`set of`**'. Si osservi l'esempio seguente:

```
type Settimana = (LUNEDÌ, MARTEDÌ, MERCOLEDÌ,
                  GIOVEDÌ, VENERDÌ, SABATO, DOMENICA);
…
type Lavoro    = set of Settimana;
…
var tutti     : Lavoro;
    presenze  : Lavoro;
    assenze   : Lavoro;
    altri     : Lavoro;
…
```

Le variabili '**`tutti`**', '**`presenze`**' e '**`assenze`**', definite del tipo '**`Lavoro`**', il quale a sua volta è definito come insieme di tutti i simboli del tipo '**`Settimana`**', possono contenere un sottoinsieme di tali simboli.

```
…
begin
    …
    presenze := (LUNEDÌ, MERCOLEDÌ, VENERDÌ,
                 DOMENICA);
    …
    tutti := (LUNEDÌ..DOMENICA);
    …
    assenze := tutti - presenze;
    …
    altri := assenze;
    …
    tutti := assenze + presenze;
    …
end.
```

L'esempio mostra alcuni modi in cui possono essere utilizzate le variabili contenenti insiemi e quali espressioni si possono realizzare. In pratica:

- due variabili dello stesso tipo di insieme possono essere assegnate l'una nell'altra;
- due variabili dello stesso tipo di insieme possono essere sommate, generando un insieme risultato dell'unione dei due;
- tra due variabili dello stesso tipo di insieme può essere indicata una sottrazione, con la quale si genera un insieme risultato dall'eliminazione degli elementi presenti nella seconda variabile.

A parte gli assegnamenti che possono essere fatti alle variabili contenenti un insieme, è poi necessario poter verificare il contenuto di tali variabili, con istruzioni apposite. Per questo si usa la parola chiave '**`in`**'. L'esempio seguente dovrebbe essere autoesplicativo.

```
    if LUNEDÌ in presenze then begin
        …
        …
    end;
    if MARTEDÌ in presenze then begin
        …
        …
    end;
```

Un insieme può essere definito anche come gruppo di valori di un intervallo, come nell'esempio seguente in cui si definisce un tipo nuovo che rappresenta l'insieme delle lettere minuscole.

```
type Lettere    = set of 'a'..'z';
```

Nello stesso modo, si può utilizzare la parola chiave '**`in`**' per verificare che un valore appartenga a un insieme definito in forma di intervallo.

```
if iniziale in 'a'..'z' then begin
    …
end;
```

Record

Il record è un tipo di dati composto dall'insieme di altri tipi, ognuno con una sua denominazione. L'esempio seguente mostra in che modo possano essere creati tipi nuovi definiti come record.

```
type Datario =
    record
        anno    : integer;
        mese    : integer;
        giorno  : integer;
    end;

type Anagrafico =
    record
        cognome : array[1..40] of char;
        nome    : array[1..40] of char;
        luogo   : array[1..40] of char;
        data    : Datario;
    end;
```

L'esempio vuole mostrare la creazione di un record anagrafico con tutti i dati (riferiti alla nascita) che permettono di identificare una persona. Si può osservare che la data (di nascita) è stata definita come tipo '**Datario**', che a sua volta è un altro record.

Quando si dichiara una variabile come tipo record, si pone il problema di accedere ai vari elementi di questo. Per farlo si usa l'operatore punto ('**.**'). Si osservi l'esempio seguente, in cui si dichiara un array di dati anagrafici e quindi si assegnano i valori per il primo elemento di questo array.

```
…
var anagrafe : array[1..10] of Anagrafico;
…
begin
    …
    anagrafe[1].cognome         := 'Pallino';
    anagrafe[1].nome            := 'Pinco';
    anagrafe[1].luogo           := 'Sferopoli';
    anagrafe[1].data.anno       := 1990;
    anagrafe[1].data.mese       := 1;
    anagrafe[1].data.giorno     := 31;
    …
end;
```

Come si può osservare, per inserire le informazioni sulla data di nascita, è stato necessario usare due volte il punto per accedere agli elementi del sottorecord '**data**'.

Una variabile definita come record può ricevere l'assegnamento in blocco di un'altra variabile record, purché dello stesso tipo.

With

Quando si utilizzano frequentemente i record, potrebbe essere conveniente specificare che in una porzione di codice sorgente si vuole fare riferimento a elementi di una variabile determinata. Si osservi l'esempio seguente, che è una variante di quanto già visto in precedenza.

```
…
var anagrafe : array[1..10] of Anagrafico;
…
begin
    …
    with anagrafe[1] do begin
        cognome         := 'Pallino';
        nome            := 'Pinco';
        luogo           := 'Sferopoli';
        data.anno       := 1990;
        data.mese       := 1;
        data.giorno     := 31;
    end;
    …
end;
```

Il significato dovrebbe essere evidente: nell'intervallo delimitato dalle parole chiave '**begin**' '**end**', tutti i nomi si riferiscono a elementi di '**anagrafe[1]**'.

Riferimenti

- Gordon Dodrill, *Pascal Language Tutorial*
  http://www.geocities.com/hotdogcom/ptutor/paslist.html
  http://packetstormsecurity.nl/programming-tutorials/Pascal/pascal-tutorial/paslist.htm

Pascal: esempi di programmazione

Problemi elementari di programmazione 979
Somma tra due numeri positivi 979
Moltiplicazione di due numeri positivi attraverso la somma 980
Divisione intera tra due numeri positivi 981
Elevamento a potenza 981
Radice quadrata 982
Fattoriale 983
Massimo comune divisore 984
Numero primo 984
Scansione di array 985
Ricerca sequenziale 985
Ricerca binaria 986
Algoritmi tradizionali 987
Bubblesort 987
Torre di Hanoi 988
Quicksort 989
Permutazioni 991

Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Lo scopo di questi esempi è solo didattico, utilizzando forme non ottimizzate per la velocità di esecuzione.

Problemi elementari di programmazione 979
Somma tra due numeri positivi 979
Moltiplicazione di due numeri positivi attraverso la somma 980
Divisione intera tra due numeri positivi 981
Elevamento a potenza 981
Radice quadrata 982
Fattoriale 983
Massimo comune divisore 984
Numero primo 984
Scansione di array 985
Ricerca sequenziale 985
Ricerca binaria 986
Algoritmi tradizionali 987
Bubblesort 987
Torre di Hanoi 988
Quicksort 989
Permutazioni 991

Problemi elementari di programmazione

In questa sezione vengono mostrati alcuni algoritmi elementari portati in Pascal.

Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
(* ============================================================ *)
(* Somma.pas                                                    *)
(* Somma esclusivamente valori positivi.                        *)
(* ============================================================ *)
program Sommare;

var     x       : integer;
        y       : integer;
        z       : integer;

(* ============================================================ *)
(* somma( <x>, <y> )                                            *)
(* ------------------------------------------------------------ *)
```

```
function somma( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := x;

    for i := 1 to y do begin
        z := z+1;
    end;

    somma := z;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := somma( x, y );

    Write( x, ' + ', y, ' = ', z );

end.

(* ================================================================ *)
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
function somma( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := x;
    i := 1;

    while i <= y do begin
        z := z+1;
        i := i+1;
    end;

    somma := z;

end;
```

Moltiplicazione di due numeri positivi attraverso la somma

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
(* ================================================================ *)
(* Moltiplica.pas                                                   *)
(* Moltiplica esclusivamente valori positivi.                       *)
(* ================================================================ *)
program Moltiplicare;

var     x       : integer;
        y       : integer;
        z       : integer;

(* ================================================================ *)
(* moltiplica( <x>, <y> )                                           *)
(* ---------------------------------------------------------------- *)
function moltiplica( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := 0;

    for i := 1 to y do begin
        z := z+x;
    end;

    moltiplica := z;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );
```



```
    z := moltiplica( x, y );

    Write( x, ' * ', y, ' = ', z );

end.

(* ================================================================ *)
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
function moltiplica( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := 0;
    i := 1;

    while i <= y do begin
        z := z+x;
        i := i+1;
    end;

    moltiplica := z;

end;
```

Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
(* ================================================================ *)
(* Dividi.pas                                                       *)
(* Divide esclusivamente valori positivi.                           *)
(* ================================================================ *)
program Dividere;

var     x       : integer;
        y       : integer;
        z       : integer;

(* ================================================================ *)
(* dividi( <x>, <y> )                                               *)
(* ---------------------------------------------------------------- *)
function dividi( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := 0;
    i := x;

    while i >= y do begin
        i := i - y;
        z := z+1;
    end;

    dividi := z;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := dividi( x, y );

    Write( x, ' / ', y, ' = ', z );

end.
(* ================================================================ *)
```

Elevamento a potenza

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
(* ================================================================ *)
(* Exp.pas                                                          *)
(* Eleva a potenza.                                                 *)
(* ================================================================ *)
program Potenza;

var     x       : integer;
        y       : integer;
        z       : integer;
```

```
(* ================================================================ *)
(* exp( <x>, <y> )                                                  *)
(* ---------------------------------------------------------------- *)
function exp( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := 1;

    for i := 1 to y do begin
        z := z * x;
    end;

    exp := z;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := exp( x, y );

    Write( x, ' ** ', y, ' = ', z );

end.
(* ================================================================ *)
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
(* ================================================================ *)
(* exp( <x>, <y> )                                                  *)
(* ---------------------------------------------------------------- *)
function exp( x : integer; y : integer ) : integer;

var     z       : integer;
        i       : integer;

begin

    z := 1;
    i := 1;

    while i <= y do begin
        z := z * x;
        i := i+1;
    end;

    exp := z;

end;
```

È possibile usare anche un algoritmo ricorsivo:

```
function exp( x : integer; y : integer ) : integer;

begin

    if x = 0 then
        begin
            exp := 0;
        end
    else if y = 0 then
        begin
            exp := 1;
        end
    else
        begin
            exp := ( x * exp(x, y-1) );
        end
    ;

end;
```

Radice quadrata

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
(* ================================================================ *)
(* Radice.pas                                                       *)
(* Radice quadrata.                                                 *)
(* ================================================================ *)
program RadiceQuadrata;

var     x       : integer;
        z       : integer;

(* ================================================================ *)
(* radice( <x> )                                                    *)
(* ---------------------------------------------------------------- *)
function radice( x : integer; ) : integer;

var     z       : integer;
        t       : integer;
        ciclo   : boolean;

begin

    z := 0;
    t := 0;
    ciclo := TRUE;

    while ciclo do begin

        t := z * z;

        if t > x then
            begin
                z := z-1;
                radice := z;
                ciclo := FALSE;
            end
        ;

        z := z+1;

    end;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il numero intero positivo: ' );
    Readln( x );

    z := radice( x );

    Writeln( 'La radice di ', x, ' e'' ', z );

end.
(* ================================================================ *)
```

Fattoriale

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
(* ================================================================ *)
(* Fact.pas                                                         *)
(* Fattoriale.                                                      *)
(* ================================================================ *)
program Fattoriale;

var     x       : integer;
        z       : integer;

(* ================================================================ *)
(* fact( <x> )                                                      *)
(* ---------------------------------------------------------------- *)
function fact( x : integer ) : integer;

var     i       : integer;

begin

    i := x - 1;

    while i > 0 do begin

        x := x * i;
        i := i-1;

    end;

    fact := x;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il numero intero positivo: ' );
    Readln( x );

    z := fact( x );

    Writeln( 'Il fattoriale di ', x, ' e'' ', z );

end.

(* ================================================================ *)
```

In alternativa, l'algoritmo si può tradurre in modo ricorsivo:

```
function fact( x : integer ) : integer;

begin

    if x > 1 then
        begin
            fact := ( x * fact( x - 1 ) )
        end
    else
        begin
            fact := 1
        end
    ;

end;
```

Massimo comune divisore

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione 62.3.7.

```
(* ================================================================ *)
(* MCD.pas                                                          *)
(* Massimo Comune Divisore.                                         *)
(* ================================================================ *)
program MassimoComuneDivisore;

var     x       : integer;
        y       : integer;
        z       : integer;

(* ================================================================ *)
(* mcd( <x>, <y> )                                                  *)
(* ---------------------------------------------------------------- *)
function mcd( x : integer; y : integer ) : integer;

begin

    while x <> y do begin

        if x > y then
            begin
                x := x - y;
            end
        else
            begin
                y := y - x;
            end
        ;

    end;

    mcd := x;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il primo numero intero positivo: ' );
    Readln( x );
    Write( 'Inserisci il secondo numero intero positivo: ' );
    Readln( y );

    z := mcd( x, y );

    Write( 'Il massimo comune divisore tra ', x, ' e ', y, ' e'' ', z );

end.

(* ================================================================ *)
```

Numero primo

Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione 62.3.8.

```
(* ================================================================ *)
(* Primo.pas                                                        *)
(* ================================================================ *)
program NumeroPrimo;

var     x       : integer;

(* ================================================================ *)
(* primo( <x> )                                                     *)
(* ---------------------------------------------------------------- *)
function primo( x : integer ) : boolean;

var     np      : boolean;
        i       : integer;
        j       : integer;

begin

    np := TRUE;
    i := 2;

    while (i < x) AND np do begin
```



```
        j := x / i;
        j := x - (j * i);

        if j = 0  then
            begin
                np := FALSE;
            end
        else
            begin
                i := i+1;
            end
        ;

    end;

    primo := np;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci un numero intero positivo: ' );
    Readln( x );

    if primo( x ) then
        begin
            Writeln( 'E'' un numero primo' );
        end
    else
        begin
            Writeln( 'Non e'' un numero primo' );
        end
    ;

end.
(* ================================================================ *)
```

Scansione di array

In questa sezione vengono mostrati alcuni algoritmi, legati alla scansione degli array, portati in Pascal.

Per semplicità, gli esempi mostrati fanno uso di array dichiarati globalmente, che come tali sono accessibili alle procedure e alle funzioni senza necessità di farne riferimento all'interno delle chiamate.

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
(* ================================================================ *)
(* RicercaSeq.pas                                                   *)
(* Ricerca sequenziale.                                             *)
(* ================================================================ *)
program RicercaSequenziale;

const   DIM     = 100;

var     lista   : array[1..DIM] of integer;
        x       : integer;
        i       : integer;
        z       : integer;

(* ================================================================ *)
(* ricercaseq( <x>, <ele-inf>, <ele-sup> )                          *)
(* ---------------------------------------------------------------- *)
function ricercaseq( x : integer; a : integer; z : integer ) : integer;

var     i       : integer;

begin

    (* ------------------------------------------------------------ *)
    (* Se l'elemento non viene trovato, il valore -1 segnala        *)
    (* l'errore.                                                    *)
    (* ------------------------------------------------------------ *)
    ricercaseq := -1;

    (* ------------------------------------------------------------ *)
    (* Scandisce l'array alla ricerca dell'elemento.                *)
    (* ------------------------------------------------------------ *)
    for i := a to z do begin

        if x = lista[i] then
            begin
                ricercaseq := i;
            end
        ;

    end;

end;
```

```
(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    Writeln( 'Inserire il valore da cercare' );
    Readln( x );

    i := ricercaseq( x, 1, z );

    Writeln( 'Il valore cercato si trova nell''elemento', i );

end.
(* ================================================================ *)
```

Esiste anche una soluzione ricorsiva che viene mostrata nella subroutine seguente:

```
function ricercaseq( x : integer; a : integer; z : integer ) : integer;

begin

    if a > z then
        begin

            (* ------------------------------------------------- *)
            (* La corrispondenza non è stata trovata.            *)
            (* ------------------------------------------------- *)
            ricercaseq := -1;
        end
    else if x = lista[a] then
        begin
            ricercaseq := a;
        end
    else
        begin
            ricercaseq := ricercaseq( x, a+1, z);
        end
    ;

end;
```

Ricerca binaria

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione 62.4.2.

```
(* ================================================================ *)
(* RicercaBin.pas                                                   *)
(* Ricerca binaria.                                                 *)
(* ================================================================ *)
program RicercaBinaria;

const   DIM     = 100;

var     lista   : array[1..DIM] of integer;
        x       : integer;
        i       : integer;
        z       : integer;

(* ================================================================ *)
(* ricercabin( <x>, <ele-inf>, <ele-sup> )                          *)
(* ---------------------------------------------------------------- *)
function ricercabin( x : integer; a : integer; z : integer ) : integer;

var     m       : integer;

begin

    (* ------------------------------------------------------------ *)
    (* Determina l'elemento centrale.                               *)
    (* ------------------------------------------------------------ *)
    m := ( a + z ) / 2;

    if m < a then
        begin

            (* --------------------------------------------------- *)
            (* Non restano elementi da controllare.                *)
            (* --------------------------------------------------- *)
            ricercabin := -1;
        end
    else if x < lista[m] then
        begin

            (* --------------------------------------------------- *)
            (* Si ripete la ricerca nella parte inferiore.         *)
            (* --------------------------------------------------- *)
            ricercabin := ricercabin( x, a, m-1 );
        end
    else if x > lista[m] then
        begin

            (* --------------------------------------------------- *)
            (* Si ripete la ricerca nella parte superiore.         *)
            (* --------------------------------------------------- *)
            ricercabin := ricercabin( x, m+1, z );
        end
    else
        begin

            (* --------------------------------------------------- *)
            (* m rappresenta l'indice dell'elemento cercato.       *)
            (* --------------------------------------------------- *)
            ricercabin := m;
        end
    ;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    Writeln( 'Inserire il valore da cercare' );
    Readln( x );

    i := ricercabin( x, 1, z );

    Writeln( 'Il valore cercato si trova nell''elemento', i );

end.
(* ================================================================ *)
```

Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in Pascal.

Bubblesort

Il problema del Bubblesort è stato descritto nella sezione 62.5.1. Viene mostrata prima una soluzione iterativa e successivamente la funzione '**bsort**' in versione ricorsiva.

```
(* ================================================================ *)
(* BSort.pas                                                        *)
(* ================================================================ *)
program BubbleSort;

const   DIM     = 100;

var     lista   : array[1..DIM] of integer;
        i       : integer;
        z       : integer;

(* ================================================================ *)
(* bsort( <ele-inf>, <ele-sup> )                                    *)
(* ---------------------------------------------------------------- *)
procedure bsort( a : integer; z : integer );

var     scambio : integer;
        j       : integer;
        k       : integer;

begin

    (* ------------------------------------------------------------ *)
    (* Inizia il ciclo di scansione dell'array.                     *)
    (* ------------------------------------------------------------ *)
    for j := a to ( z-1 ) do begin

        (* --------------------------------------------------------- *)
        (* Scansione interna dell'array per collocare nella          *)
        (* posizione j l'elemento giusto.                            *)
        (* --------------------------------------------------------- *)
```

```
        for k := ( j+1 ) to z do begin

            if lista[k] < lista[j] then
                begin

                    (* ----------------------------------------- *)
                    (* Scambia i valori.                         *)
                    (* ----------------------------------------- *)
                    scambio := lista[k];
                    lista[k] := lista[j];
                    lista[j] := scambio;
                end
            ;

        end;
    end;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    bsort( 1, z );

    Writeln( 'Array ordinato:' );

    for i := 1 to z do begin
        Write( lista[i] );
    end;

end.
(* ================================================================ *)
```

Segue la procedura '**`bsort`**' in versione ricorsiva:

```
procedure bsort( a : integer; z : integer );

var     scambio : integer;
        k       : integer;

begin

    if a < z then
        begin

            (* ------------------------------------------------- *)
            (* Scansione interna dell'array per collocare nella  *)
            (* posizione j l'elemento giusto.                    *)
            (* ------------------------------------------------- *)
            for k := ( a+1 ) to z do begin

                if lista[k] < lista[a] then
                    begin

                        (* ----------------------------------- *)
                        (* Scambia i valori.                   *)
                        (* ----------------------------------- *)
                        scambio := lista[k];
                        lista[k] := lista[a];
                        lista[a] := scambio;
                    end
                ;

            end;

            bsort( a+1, z );

        end
    ;

end;
```

Torre di Hanoi

«

Il problema della torre di Hanoi è descritto nella sezione 62.5.3.

```
(* ================================================================ *)
(* Hanoi.pas                                                        *)
(* Torre di Hanoi.                                                  *)
(* ================================================================ *)
program TorreHanoi;

var     n       : integer;
        p1      : integer;
        p2      : integer;

(* ================================================================ *)
(* hanoi( <n>, <p1>, <p2> )                                         *)
(* ---------------------------------------------------------------- *)
procedure hanoi( n : integer; p1 : integer; p2 : integer );

begin

    if n > 0 then
        begin
            hanoi( n-1, p1, 6-p1-p2 );

            Writeln(
                'Muovi l''anello ', n:1,
                ' dal piolo ', p1:1,
                ' al piolo ', p2:1
            );

            hanoi( n-1, 6-p1-p2, p2 );
        end
    ;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln;
    Write( 'Inserisci il numero di anelli: ' );
    Readln( n );
    Write( 'Inserisci il piolo iniziale: ' );
    Readln( p1 );
    Write( 'Inserisci il piolo finale: ' );
    Readln( p2 );

    hanoi( n, p1, p2 );

end.
(* ================================================================ *)
```

Quicksort

«

L'algoritmo del Quicksort è stato descritto nella sezione 62.5.4.

```
(* ================================================================ *)
(* QSort.pas                                                        *)
(* ================================================================ *)
program QuickSort;

const   DIM     = 100;

var     lista   : array[1..DIM] of integer;
        i       : integer;
        z       : integer;

(* ================================================================ *)
(* part( <ele-inf>, <ele-sup> )                                     *)
(* ---------------------------------------------------------------- *)
function part( a : integer; z : integer ) : integer;

var     scambio : integer;
        i       : integer;
        cf      : integer;
        loop1   : boolean;
        loop2   : boolean;
        loop3   : boolean;

begin

    (* ------------------------------------------------------------ *)
    (* Si assume che a sia inferiore a z.                           *)
    (* ------------------------------------------------------------ *)
    i := a+1;
    cf := z;

    (* ------------------------------------------------------------ *)
    (* Inizia il ciclo di scansione dell'array.                     *)
    (* ------------------------------------------------------------ *)
    loop1 := TRUE;
    while loop1 do begin

        loop2 := TRUE;
        while loop2 do begin

            (* --------------------------------------------- *)
            (* Sposta i a destra.                            *)
            (* --------------------------------------------- *)
            if ( lista[i] > lista[a] ) OR ( i >= cf ) then
                begin
                    loop2 := FALSE;
                end
            else
                begin
                    i := i+1;
                end
            ;

        end;
```

```
        loop3 := TRUE;
        while loop3 do begin

            (* ------------------------------------------------- *)
            (* Sposta cf a sinistra.                             *)
            (* ------------------------------------------------- *)
            if lista[cf] <= lista[a] then
                begin
                    loop3 := FALSE;
                end
            else
                begin
                    cf := cf-1;
                end
            ;

        end;

        if cf <= i then
            begin

                (* ------------------------------------------------- *)
                (* è avvenuto l'incontro tra i e cf.                 *)
                (* ------------------------------------------------- *)
                loop1 := FALSE;
            end
        else
            begin

                (* ------------------------------------------------- *)
                (* Vengono scambiati i valori.                       *)
                (* ------------------------------------------------- *)
                scambio := lista[cf];
                lista[cf] := lista[i];
                lista[i] := scambio;

                i := i+1;
                cf := cf-1;
            end
        ;
    end;

    (* ------------------------------------------------------------ *)
    (* A questo punto, lista[a..z] è stata ripartita e cf è la      *)
    (* collocazione finale.                                         *)
    (* ------------------------------------------------------------ *)
    scambio := lista[cf];
    lista[cf] := lista[a];
    lista[a] := scambio;

    (* ------------------------------------------------------------ *)
    (* In questo momento, lista[cf] è un elemento (un valore) nella *)
    (* posizione giusta.                                            *)
    (* ------------------------------------------------------------ *)
    part := cf

end;

(* ================================================================ *)
(* quicksort( <ele-inf>, <ele-sup> )                                *)
(* ---------------------------------------------------------------- *)
procedure quicksort( a : integer; z : integer );

var     cf      : integer;

begin

    if z > a then
        begin
            cf := part( a, z );
            quicksort( a, cf-1 );
            quicksort( cf+1, z );
        end
    ;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    quicksort( 1, z );

    Writeln( 'Array ordinato:' );
    for i := 1 to z do begin
        Write( lista[i] );
    end;

end.
(* ================================================================ *)
```


Permutazioni

L'algoritmo ricorsivo delle permutazioni è descritto nella sezione 62.5.5.

```
(* ================================================================ *)
(* Permuta.pas                                                      *)
(* ================================================================ *)
program Permutazioni;

const   DIM     = 100;

var     lista   : array[1..DIM] of integer;
        i       : integer;
        z       : integer;

(* ================================================================ *)
(* permuta( <ele-inf>, <ele-sup>, <elementi-totali> )               *)
(* ---------------------------------------------------------------- *)
function permuta( a : integer; z : integer; elementi : integer ) : integer;

var     scambio : integer;
        k       : integer;
        i       : integer;

begin

    (* ------------------------------------------------------------ *)
    (* Se il segmento di array contiene almeno due elementi,        *)
    (* si procede.                                                  *)
    (* ------------------------------------------------------------ *)
    if ( z-a ) >= 1 then
        begin

            (* ------------------------------------------------- *)
            (* Inizia il ciclo di scambi tra l'ultimo elemento e *)
            (* uno degli altri contenuti nel segmento di array.  *)
            (* ------------------------------------------------- *)
            k := z;
            while k >= a do begin

                (* ------------------------------------------- *)
                (* Scambia i valori.                           *)
                (* ------------------------------------------- *)
                scambio := lista[k];
                lista[k] := lista[z];
                lista[z] := scambio;

                (* ------------------------------------------- *)
                (* Esegue una chiamata ricorsiva per permutare un *)
                (* segmento più piccolo dell'array.            *)
                (* ------------------------------------------- *)
                permuta( a, z-1, elementi );

                (* ------------------------------------------- *)
                (* Scambia i valori.                           *)
                (* ------------------------------------------- *)
                scambio := lista[k];
                lista[k] := lista[z];
                lista[z] := scambio;

                k := k-1;

            end;
        end
    else
        begin

            (* ------------------------------------------------- *)
            (* Visualizza la situazione attuale dell'array.      *)
            (* ------------------------------------------------- *)
            for i := 1 to elementi do begin
                Write( lista[i]:4 );
            end;
            Writeln;

        end
    ;

end;

(* ================================================================ *)
(* Inizio del programma.                                            *)
(* ---------------------------------------------------------------- *)
begin

    Writeln( 'Inserire il numero di elementi.' );
    Writeln( DIM, ' al massimo.' );
    Readln( z );

    if z > DIM then
        begin
            z := DIM;
        end
```

```
    ;

    Writeln( 'Inserire i valori dell''array' );

    for i := 1 to z do begin
        Write( 'elemento ', i:2, ': ' );
        Readln( lista[i] );
    end;

    permuta( 1, z, z );

end.

(* ============================================================ *)
```


Parte xxviii

Java

Java: preparazione 995
- Kaffe 995
- Kernel Linux 996
- Applet 998
- JDK 998
- GCJ 999
- Riferimenti 1000

Java: introduzione 1001
- Struttura fondamentale 1001
- Variabili e tipi di dati 1004
- Operatori ed espressioni 1005
- Strutture di controllo del flusso 1007
- Array e stringhe 1010
- Metodo «main()» 1012

Java: programmazione a oggetti 1013
- Creazione e distruzione di un oggetto 1013
- Classi 1015
- Sottoclassi 1018
- Interfacce 1019
- Pacchetti di classi 1020
- Esempi 1022

Java: esempi di programmazione 1025
- Problemi elementari di programmazione 1025
- Scansione di array 1030
- Algoritmi tradizionali 1032

Kaffe 995
Classi 995
Configurazione 995
Compilazione 996
Esecuzione 996
Kernel Linux 996
Applet 998
Verifica del funzionamento 998
JDK 998
GCJ 999
Riferimenti 1000

Java è un linguaggio di programmazione realizzato da Sun Microsystems, utilizzato originariamente per l'inserzione di programmi all'interno di pagine HTML (applet), un po' come si fa con le immagini. Per questo motivo, il risultato consueto della compilazione di un sorgente Java è una codifica intermedia, indipendente dalla piattaforma, che deve poi essere interpretata localmente dal navigatore o da un altro programma indipendente. Tuttavia, nel tempo sono stati sviluppati anche compilatori alternativi, che producono un programma eseguibile tradizionale (dipendente dalla piattaforma hardware-software).

Per programmare in Java occorre un compilatore, generalmente noto come '**`javac`**', che sia in grado di generare il formato binario Java, il cosiddetto Java bytecode. Il file che si ottiene non è propriamente un eseguibile, in quanto necessita di un interprete che generalmente è il programma '**`java`**'.

Esiste una versione ufficiale di questi strumenti, definita JDK (*Java development kit*), e altre versioni indipendenti, come per esempio Kaffe.

Nel capitolo viene descritto in particolare come utilizzare Kaffe. Alla fine del capitolo si trova la descrizione dell'installazione e della configurazione di JDK originale, oltre a una sezione sull'uso di GCJ per la compilazione di sorgenti o binari Java nel formato eseguibile adatto alla propria architettura.

Kaffe

Kaffe [1] è un compilatore di sorgenti Java e un interprete di compilati in formato Java (Java bytecode). Attualmente, si tratta di un pacchetto standard delle distribuzioni GNU, per cui non ci dovrebbero essere problemi nella sua installazione. Attualmente, assieme al compilatore e all'interprete, dovrebbero essere disponibili anche le ***classi***, ovvero le librerie Java.

Classi

Le classi di Kaffe, che ormai accompagnano questo applicativo, dovrebbero essere contenute in un solo file compresso, che deve rimanere tale. Potrebbe trattarsi di '`/usr/share/kaffe/Klasses.jar`'.

Configurazione

Se si installa Kaffe autonomamente, senza affidarsi a un pacchetto già predisposto per la propria distribuzione GNU, potrebbe essere necessario definire alcune variabili di ambiente. Nell'esempio seguente si fa riferimento a uno script per una shell Bourne o derivata:

```
CLASSPATH=.:/usr/share/kaffe/Klasses.jar
KAFFEHOME=/usr/share/kaffe
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib
export CLASSPATH
export KAFFEHOME
export LD_LIBRARY_PATH
```

Se Kaffe fosse stato installato a partire dalla directory '`/usr/local/`', si dovrebbe usare la definizione seguente:

```
CLASSPATH=.:/usr/local/share/kaffe/Klasses.jar
KAFFEHOME=/usr/local/share/kaffe
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib
export CLASSPATH
export KAFFEHOME
export LD_LIBRARY_PATH
```

Merita un po' di attenzione la variabile '**`LD_LIBRARY_PATH`**' che potrebbe essere utilizzata anche da altri programmi. '**`LD_LIBRARY_PATH`**' deve contenere i percorsi in cui si trovano i file di libreria; se il proprio sistema utilizza applicazioni che collocano le proprie librerie all'interno di directory inconsuete, queste devono essere aggiunte all'elenco. Segue un esempio esplicativo:

```
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib:/opt/mio_prog/lib:/opt/tuo_prog/lib
```

Compilazione

Per verificare che la compilazione funzioni correttamente, basta preparare il solito programma banale che visualizza un messaggio attraverso lo standard output e poi termina:

```
class CiaoMondoApp
{
    public static void main (String[] args)
      {
        System.out.println ("Ciao Mondo!");
      }
}
```

Il file deve essere salvato con il nome '`CiaoMondoApp.java`'. Kaffe, tra le altre cose, fornisce un collegamento simbolico, denominato '**`javac`**', attraverso cui avviare la compilazione. Così la compilazione avviene nello stesso modo degli strumenti JDK originali:

```
$ javac CiaoMondoApp.java [Invio]
```

Se la sintassi del sorgente Java è corretta, si ottiene un file in formato binario Java, denominato '`CiaoMondoApp.class`'.

Esecuzione

Per eseguire il binario Java generato, ovvero il file '`.class`', occorre un interprete. In questo senso, il binario Java non ha bisogno necessariamente dei permessi di esecuzione, perché viene solo letto dall'interprete.

```
$ kaffe CiaoMondoApp [Invio]

Ciao Mondo!
```

Come si può osservare dalla riga di comando, il file binario Java deve essere indicato senza l'estensione, che di conseguenza è obbligatoriamente '`.class`'. Kaffe si compone anche dello script '**`java`**', il cui scopo è quello di rendere il comando di interpretazione conforme al JDK; in pratica, '**`java`**' si limita ad avviare il comando '**`kaffe`**'.

```
$ java CiaoMondoApp [Invio]
```

Tuttavia, questo script potrebbe essere modificato in modo da permettere l'avvio di un eseguibile Java anche se è stato fornito il nome del file corrispondente, completo di estensione '`.class`'. L'esempio seguente rappresenta le modifiche che potrebbero essere apportate in tal senso:

```
#!/bin/sh
#
# /usr/bin/java

CLASSE=`/bin/basename $1 .class`
shift
kaffe $CLASSE $@
```

Kernel Linux

Come è noto, uno script viene interpretato automaticamente in base alla convenzione per cui la prima riga inizia con l'indicazione del programma adatto. Per esempio: '**`#!/bin/sh`**', '**`#!/bin/bash`**' e '**`#!/usr/bin/perl`**'. Con i binari Java ciò non è possibile, quindi, per ottenere l'avvio automatico dell'interprete '**`java`**', occorre che il kernel ne sia informato. Per la precisione, occorre attivare la funzionalità generica di riconoscimento dei binari (sezione 8.3.1); inoltre occorre accertarsi che la directory '`/proc/sys/fs/binfmt_misc/`' contenga i file '`register`' e '`status`'. Se le cose non stanno così, è necessario innestare il file system '**`binfmt_misc`**':

```
# mount -t binfmt_misc none /proc/sys/fs/binfmt_misc [Invio]
```

Una volta che sono disponibili i file virtuali '`register`' e '`status`', per attivare la funzionalità occorre intervenire con il comando seguente:

```
# echo 1 > /proc/sys/fs/binfmt_misc/status [Invio]
```

Per disattivarla, basta utilizzare il valore zero.

```
# echo 0 > /proc/sys/fs/binfmt_misc/status [Invio]
```

Quando tutto è in ordine per la gestione dei binari eterogenei, si può definire quali file devono essere riconosciuti e quali interpreti devono essere avviati di conseguenza. Nel caso dei binari Java normali, si tratta di eseguire il comando seguente (il percorso dell'interprete, '`/usr/bin/java`' può essere cambiato a seconda delle proprie necessità).

```
# echo ':Java:M::\xca\xfe\xba\xbe::/usr/bin/java:' ↩
↪> /proc/sys/fs/binfmt_misc/register [Invio]
```

In alternativa, se si è sicuri dell'estensione '`.class`', si può utilizzare il comando seguente:

```
# echo ':Java:E::class::/usr/bin/java:' ↩
↪> /proc/sys/fs/binfmt_misc/register [Invio]
```

Per verificare che la definizione sia stata recepita correttamente dal kernel, si può leggere il contenuto del file virtuale '`/proc/sys/fs/binfmt_misc/Java`', creato a seguito di uno dei due comandi mostrati sopra.

Quando il kernel è predisposto nel modo appena visto, si possono rendere eseguibili i file binari Java; così, quando si tenta di avviarli, il kernel stesso avvia invece il comando seguente:

```
java file_binario_java argomenti
```

Lo svantaggio di questo sistema sta nel fatto che il nome del file binario Java viene indicato con tutta l'estensione, cosa che normalmente crea dei problemi, sia a Kaffe che al JDK. Per questo, conviene che '`/usr/bin/java`' sia uno script predisposto per risolvere il problema, come già mostrato nella sezione precedente.

Se invece di usare Kaffe si usa il JDK originale, conviene modificare il nome dell'interprete Java, per esempio in '**`java1`**', realizzando poi un file script analogo a quello già visto.

```
#!/bin/sh
#
# /usr/bin/java

CLASSE=`/bin/basename $1 .class`
shift
java1 $CLASSE $@
```

C'è però una cosa che occorre tenere a mente. Con GNU/Linux, così come con altri sistemi, non è possibile avviare un eseguibile se il nome non viene indicato per esteso. In pratica, non è pensabile che succeda quanto accade in Dos in cui i file che finiscono per '`.COM`' o '`.EXE`' sono avviati semplicemente nominandoli senza estensione.

Per chi ha usato GNU/Linux da un po' di tempo ciò dovrebbe essere logico, ma con Java si rischia ancora di essere ingannati: il fatto che, sia l'interprete '**`java`**' originale, sia '**`kaffe`**', vogliano il nome dell'eseguibile Java senza l'estensione '`.class`', non deve fare supporre che ciò valga anche per il kernel. Per cui, se si avvia '`CiaoMondoApp.class`' nel modo seguente,

```
$ java CiaoMondoApp [Invio]
```

quando si vuole che sia il kernel a fare tutto questo in modo automatico, il comando diviene il seguente:

```
$ CiaoMondoApp.class [Invio]
```

Se si tentasse si eseguire il comando seguente, si otterrebbe una segnalazione di errore del tipo: '**`command not found`**'.

```
$ CiaoMondoApp [Invio]
```

Applet

Un'applet Java è un programma particolare che può essere incorporato in un documento HTML. Il meccanismo è simile all'inserzione di immagini; l'effetto è quello di un programma grafico che, invece di utilizzare una finestra si inserisce in un'area prestabilita del documento HTML. Un'applet Java non può quindi vivere da sola, richiede sempre l'abbinamento a una pagina HTML.

Il modo migliore per vedere il funzionamento di un programma del genere è attraverso l'utilizzo di un navigatore in grado di eseguire tali applet.

Verifica del funzionamento

Per verificare il funzionamento di un'applet si può provare il solito programma banale. In questo caso si comincia con la realizzazione di una pagina HTML che incorpori l'applet che si vuole realizzare.

```
<!-- CiaoMondo.html -->
<HTML>
<HEAD>
    <TITLE>La mia prima applet</TITLE>
</HEAD>
<BODY>
<OBJECT CODETYPE="application/java"
    CLASSID="java:CiaoMondo.class">
Applet Java
</OBJECT>
</BODY>
</HTML>
```

Come si vede, l'elemento '**`OBJECT`**' dichiara l'utilizzo dell'applet '`CiaoMondo.class`'. Segue il sorgente dell'applet:

```
// CiaoMondo.java

import java.applet.Applet;
import java.awt.Graphics;

public class CiaoMondo extends Applet
{
    public void paint (Graphics g)
    {
        g.drawString ("Ciao Mondo!", 50, 25);
    }
}
```

Si compila il sorgente '`CiaoMondo.java`' nel solito modo, ottenendo il binario Java '`CiaoMondo.class`'

```
$ javac CiaoMondo.java [Invio]
```

Quando si carica il file '`CiaoMondo.html`' attraverso un navigatore adatto, incontrando l'elemento '**`OBJECT`**' che fa riferimento al binario Java '**`CiaoMondo.class`**', viene caricato il programma '`CiaoMondo.class`' nell'area stabilita.

All'interno di quell'area, a partire dall'angolo superiore sinistro, vengono calcolate le coordinate (x=50, y=25) dell'istruzione '**`g.drawString("Ciao mondo!", 50, 25)`**' vista nell'applet.

JDK

software non libero

JDK [2] è il pacchetto originale per la compilazione e l'esecuzione di applicativi Java. Viene distribuito in forma binaria, già compilata. Per ottenerlo, si può consultare *http://www.blackdown.org/* o eventualmente si può fare una ricerca attraverso *http://www.google.com* per i file contenenti la stringa '`linux-jdk`' (si potrebbero trovare nomi come '`linux-jdk.1.1.3-v2.tar.gz`'). Se si desidera installare il JDK è importante verificare di non avere tracce di Kaffe.

Il JDK può essere installato a partire da qualunque punto del proprio file system. Qui viene proposta l'installazione a partire da '`/opt/`'.

Se nel proprio sistema non è presente, la si può creare, quindi al suo interno si può espandere il contenuto del pacchetto JDK. Si ottiene così la directory '`jdk`***`versione`***`/`', per esempio '`jdk1.1.3/`'. Per motivi pratici è opportuno modificare il nome della directory, o creare un collegamento simbolico, in modo che vi si possa accedere utilizzando il nome '`/opt/java/`'.

Prima di poter funzionare, il JDK deve essere configurato attraverso delle variabili di ambiente opportune. Nell'esempio seguente si mostra un pezzo di script per una shell Bourne o derivata, in grado di predisporre le variabili necessarie:

```
PATH="/opt/java/bin:$PATH"
CLASSPATH=.:/opt/java/lib/classes.zip:/opt/java/lib/classes
JAVA_HOME=/opt/java
export PATH
export CLASSPATH
export JAVA_HOME
```

Per il funzionamento si può rivedere quanto già indicato per Kaffe. In questo caso, utilizzando il JDK originale, il compilatore è proprio '**`javac`**' e l'esecutore (o interprete) è '**`java`**'.

GCJ

GCJ [3] è un programma frontale per il controllo del compilatore GCC e di altri programmi accessori, il cui scopo è quello di compilare sorgenti Java.

La compilazione può avvenire a diversi livelli: da sorgenti Java ('`.java`') o da binari Java ('`.class`') si può arrivare a un file eseguibile per il proprio sistema operativo; in alternativa si possono semplicemente compilare dei sorgenti Java per generare i binari Java corrispondenti ('`.class`'). Semplificando le cose, si possono distinguere questi tre tipi di comandi per la compilazione:

- `gcj -C` ***file_sorgente_java*** ...

  per generare binari Java (file '`.class`');

- `gcj --main=`***classe_principale*** `-o` ***file_da_generare file_sorgente_java*** ...

  per generare un eseguibile a partire da dei sorgenti Java (file '`.java`');

- `gcj --main=`***classe_principale*** `-o` ***file_da_generare binario_java*** ...

  per generare un eseguibile a partire da binari Java (file '`.class`').

Supponendo di avere il solito esempio già visto in precedenza,

```
class CiaoMondoApp
{
    public static void main (String[] args)
    {
        System.out.println ("Ciao Mondo!");
    }
}
```

supponendo questa volta che sia contenuto nel file '`ciao_mondo.java`', si può generare il binario Java '`CiaoMondoApp.class`' con il comando seguente:

```
$ gcj -C ciao_mondo.java [Invio]
```

Per compilare il binario Java in modo da ottenere un binario adatto al sistema operativo e all'architettura del proprio elaboratore, si può usare il comando seguente, generando quindi l'eseguibile '**`ciao`**':

```
$ gcj --main=CiaoMondoApp -o ciao CiaoMondoApp.class [Invio]
```

Infine, per compilare direttamente il sorgente Java, si può agire nello stesso modo:

```
$ gcj --main=CiaoMondoApp -o ciao ciao_mondo.java [Invio]
```

GCJ riconosce la variabile di ambiente '**`CLASSPATH`**', per la ricerca delle classi, fornendo anche la possibilità di indicare tale informazione attraverso la riga di comando, con delle opzioni che qui non vengono mostrate.

Opzione	Descrizione
`-c`	In questo caso, i file in ingresso sono sorgenti Java e vengono compilati generando le classi in forma di binari Java.
`--main=`***classe***	Questa opzione permette di stabilire quale sia la classe da utilizzare come principale, in modo che il programma che si genera inizi da lì il suo funzionamento.
`-o` ***file***	Definisce il nome dell'eseguibile da generare, quando la compilazione non è destinata a ottenere soltanto un binario Java.

Riferimenti

- *TransVirtual Technologies Inc.*
  http://www.transvirtual.com
- Riferimenti per ottenere il JDK dalla rete
  http://www.blackdown.org/
- *The source for Java, Documentation*
  http://java.sun.com/docs/index.html
- *The source for Java, Tutorial*
  http://java.sun.com/docs/books/tutorial/index.html

[1] **Kaffe** software libero con licenza speciale

[2] **JDK** software non libero

[3] **GCJ** GNU GPL

Java: introduzione

Struttura fondamentale 1001
- Commenti 1002
- Nomi ed estensioni 1002
- Istruzioni 1002
- Librerie di classi 1002
- Dichiarazione della classe 1003
- Contenuto della classe 1003
- Variabili e tipi di dati 1003
- Chiamata per valore 1003

Variabili e tipi di dati 1004
- Tipi 1004
- Costanti 1004
- Campo di azione 1005

Operatori ed espressioni 1005
- Operatori aritmetici 1006
- Operatori di confronto e operatori logici 1006
- Concatenamento di stringhe 1007

Strutture di controllo del flusso 1007
- Struttura condizionale: «if» 1007
- Struttura di selezione: «switch» 1008
- Iterazione con condizione di uscita iniziale: «while» 1009
- Iterazione con condizione di uscita finale: «do-while» 1009
- Iterazione enumerativa: «for» 1009

Array e stringhe 1010
- Array 1010
- Stringhe 1011

Metodo «main()» 1012
- args 1012

Questo capitolo introduce alla programmazione in Java, in modo superficiale, per dare un'idea delle potenzialità di questo linguaggio.

Struttura fondamentale

Java è un linguaggio di programmazione strettamente OO (*Object oriented*), cioè a dire che qualunque cosa si faccia, anche un semplice programma che emette un messaggio attraverso lo standard output, va trattato secondo la programmazione a oggetti.

Ciò significa anche che i componenti di questo linguaggio hanno nomi diversi da quelli consueti. Volendo fare un abbinamento approssimativo con un linguaggio di programmazione normale, si potrebbe dire che in Java i programmi sono ***classi*** e le funzioni sono ***metodi***. Naturalmente ci sono anche tante altre cose nuove.

Fatta questa premessa, si può dare un'occhiata alla solita classe banale: quella che visualizza un messaggio e termina.

```
/**
 *  CiaoMondoApp.java
 *  La solita classe banale.
 */

import java.lang.*; // predefinita

class CiaoMondoApp
{
    public static void main (String[] args)
    {
        System.out.println ("Ciao Mondo!"); // visualizza il messaggio
    }
}
```

Il sorgente Java ha molte somiglianze con quello del linguaggio C e qui si intendono segnalare le particolarità rispetto a quel linguaggio.

Commenti

Java ammette l'uso di commenti in stile C, nella solita forma '**`/*`**…**`*/`**', ma ne introduce altri due tipi: uno per la creazione automatica di documentazione, nella forma '**`/**`**…**`*/`**', e uno per fare ignorare tutto ciò che appare a partire dal simbolo di commento fino alla fine della riga, nella forma '***`//commento`***':

```
/* commento_generico */
```

```
/** documentazione */
```

```
// commento_fino_alla_fine_della_riga
```

Tutti e tre questi tipi di commenti servono a fare ignorare al compilatore una parte del sorgente e questo dovrebbe bastare al principiante. Convenzionalmente, è conveniente usare il commento di documentazione per la spiegazione di ciò che fa la classe, all'inizio del sorgente.

Nomi ed estensioni

Le estensioni dei file Java sono definite in modo obbligatorio: '`.java`' per i sorgenti e '`.class`' per le classi (i binari Java).

Generalmente, nel sorgente, il nome della classe deve corrispondere alla radice del nome del sorgente e, di conseguenza, anche del binario Java. Per lo stile convenzionale di Java, questo nome inizia con una lettera maiuscola e non contiene simboli strani; se è composto dall'unione di più parole, ognuna di queste inizia con una lettera maiuscola.

Istruzioni

Le istruzioni seguono la convenzione del linguaggio C, per cui terminano con un punto e virgola ('**`;`**') e i raggruppamenti di queste, detti anche blocchi, si fanno utilizzando le parentesi graffe ('**`{ }`**').

```
istruzione ;
```

```
{ istruzione ; istruzione ; istruzione ; }
```

Generalmente, un'istruzione può essere interrotta e ripresa nella riga successiva, dal momento che la sua conclusione è dichiarata chiaramente dal punto e virgola finale.

Librerie di classi

Ogni programma in Java deve fare affidamento sull'utilizzo di classi fondamentali che compongono il linguaggio stesso. L'importazione delle classi necessarie viene fatta attraverso l'istruzione '**`import`**', indicando una classe particolare o un gruppo (nel secondo caso si usa un asterisco).

Nell'esempio introduttivo vengono importate tutte le classi del pacchetto '**`java.lang`**', anche se non sarebbe stato necessario dichiararlo, dato che queste classi vengono sempre importate in modo predefinito (senza di queste, nessuna classe potrebbe funzionare).

Le classi standard di Java (cioè queste librerie fondamentali), sono contenute normalmente in un archivio compresso '`.zip`', oppure '`.jar`'. Si è visto nel capitolo u122 che è importante indicare il percorso in cui si trovano, nella variabile di ambiente '**`CLASSPATH`**'.

Osservando il contenuto di questo file, si può comprendere meglio il concetto di pacchetto di classi. Segue solo un breve estratto:


```
Archive:  classes.zip
 Length     Date   Time    Name
 ------    ----    ----    ----
      0  05-19-97  22:46   java/
      0  05-19-97  22:24   java/lang/
   1322  05-19-97  22:24   java/lang/Object.class
   4202  05-19-97  22:24   java/lang/Class.class
    …
   3450  05-19-97  22:24   java/lang/System.class
    …
      0  05-19-97  22:26   java/util/
    …
      0  05-19-97  22:26   java/io/
    …
      0  05-19-97  22:42   java/awt/
    …
```

Ecco che così può diventare più chiaro il fatto che, importare tutte le classi del pacchetto '**`java.lang`**' significa in pratica includere tutte le classi contenute nella directory '`java/lang/`', anche se qui si tratta solo di un file compresso.

Dichiarazione della classe

Generalmente, un file sorgente Java contiene la dichiarazione di una sola classe, il cui nome corrisponde alla radice del file sorgente. La dichiarazione della classe delimita in pratica il contenuto del sorgente, definendo eventuali ***ereditarietà*** da altre classi esistenti.

Quando una classe non eredita da un'altra, si parla convenzionalmente di ***applicazione***, mentre quando eredita dalla classe '**`java.applet.Applet`**' (cioè da '`java/applet/Applet.class`') si usa la definizione ***applet***.

Contenuto della classe

La classe contiene essenzialmente dichiarazioni di variabili e metodi. L'esecuzione di un metodo dipende da una chiamata, detta anche ***messaggio***. Perché una classe si traduca in un programma autonomo, occorre che al suo interno ci sia un metodo che viene eseguito in modo automatico all'avvio.

Nel caso delle classi che non ereditano nulla da altre, come nell'esempio, ci deve essere il metodo '**`main`**' che viene eseguito all'avvio del binario Java contenente la classe stessa. Quando una classe eredita da un'altra, queste regole sono stabilite dalla classe ereditata.

Il metodo '**`main`**' è formato necessariamente come nell'esempio: '**`public static void main(String[] args) {...}`**'.

Variabili e tipi di dati

In Java si distinguono fondamentalmente due tipi di rappresentazione dei dati: primitivi e riferimenti a oggetti. I tipi di dati primitivi sono per esempio i soliti tipi numerici (intero, a virgola mobile, ecc.); gli altri sono ***oggetti***. Un oggetto è quindi una variabile contenente un riferimento a una struttura, più o meno complessa. In Java, gli array e le stringhe sono oggetti; pertanto non esistono tipi di dati primitivi equivalenti.

I nomi delle variabili possono essere composti utilizzando caratteri Unicode. Naturalmente, non è possibile utilizzare nomi coincidenti con parole chiave già utilizzate dal linguaggio stesso. La convenzione stilistica di Java richiede che il nome delle variabili inizi con la lettera minuscola; inoltre, se si tratta di un nome composto, la convenzione richiede di segnalare l'inizio di ogni nuova parola con una lettera maiuscola. Per esempio: '**`miaVariabile`**', '**`dataOdierna`**', '**`elencoNomiFemminili`**'.

Chiamata per valore

In Java, le chiamate dei metodi avvengono trasferendo il valore degli argomenti indicati nella chiamata stessa. Ciò significa che le modifiche che si dovessero apportare all'interno dei metodi non si riflettono all'indietro. Tuttavia, questo ragionamento vale solo per i tipi di dati primitivi, dal momento che quando si utilizzano degli oggetti, essendo questi dei riferimenti, le variazioni fatte al loro interno rimangono anche dopo la chiamata.

Variabili e tipi di dati

Si è già accennato al fatto che Java distingue tra due tipi di dati, primitivi e riferimenti a oggetti (o più semplicemente solo oggetti). L'esempio seguente mostra la dichiarazione di un intero all'interno di un metodo e il suo incremento fino a raggiungere un valore predefinito:

```
/**
 *  DieciXApp.java
 *  Un esempio di utilizzo delle variabili.
 */

import java.lang.*; // predefinita

class DieciXApp
{
    public static void main (String[] args)
    {
        int contatore = 0;

        // Inizia un ciclo in cui si emettono 10 «x» attraverso lo
        // standard output.
        while (contatore < 10)
          {
            contatore++;
            System.out.println ("x"); // emette una «x»
          }
    }
}
```

Tipi

I tipi di dati primitivi rappresentano un valore singolo. Il loro elenco si trova nella tabella u123.4.

Tabella u123.4. Elenco dei tipi di dati primitivi in Java.

Tipo	Dimensione	Descrizione
`byte`	8 bit, complemento a due.	Intero a 8 bit.
`short`	16 bit, complemento a due.	Intero ridotto.
`int`	32 bit, complemento a due.	Intero normale.
`long`	64 bit, complemento a due.	Intero molto grande.
`float`	32 bit	Virgola mobile, singola precisione.
`double`	64 bit	Virgola mobile, doppia precisione.
`char`	16 bit, carattere Unicode.	Carattere.
`boolean`	*Vero* o *Falso*.	Valore booleano.

Nell'esempio mostrato precedentemente, viene dichiarato un intero normale, '**`contatore`**', inizializzato al valore zero, che poi viene incrementato all'interno di un ciclo:

```
        int contatore = 0;

        // Inizia un ciclo in cui si emettono 10 «x» attraverso lo
        // standard output.
        while (contatore < 10)
        {
            contatore++;
            System.out.println ("x"); // emette una «x»
        }
```

Costanti

Ogni tipo primitivo ha la possibilità di essere rappresentato in forma di costante letterale. La tabella u123.6 mostra l'elenco dei tipi di dati abbinati alla rappresentazione in forma di costante letterale.

Tabella u123.6. Elenco dei tipi di dati primitivi abbinati a una possibile rappresentazione in forma di costante letterale.

Tipo	Esempio di costante	Descrizione o intervallo
`byte`	123	-128..+127
`short`	12345	-32768..+32767
`int`	1234567890	$-(2^{31})..+((2^{31})-1)$
`long`	1234567890123456789**0**	$-(2^{63})..+((2^{63})-1)$

Tipo	Esempio di costante	Descrizione o intervallo
`float`	(float)123.456	La costante con virgola è sempre a doppia precisione.
`double`	123.456	
`char`	'A'	Si usano gli apici semplici.
`boolean`	true	Si usano le parole chiave '**`true`**' e '**`false`**'.

È importante osservare che una costante numerica a virgola mobile è sempre a doppia precisione, per cui, se si vuole assegnare a una variabile a singola precisione ('**`float`**') una costante letterale, occorre una conversione di tipo, per mezzo di un cast. In seguito vengono descritte le stringhe, che si delimitano utilizzando gli apici doppi. Per ora è solo il caso ti tenere in considerazione che in Java le stringhe non sono tipi di dati primitivi, ma oggetti veri e propri.

Campo di azione

Il campo di azione delle variabili in Java viene determinato dalla posizione in cui queste vengono dichiarate. Ciò determina il momento della loro creazione e distruzione. A fianco del concetto del campo di azione, si pone quello della ***protezione***, che può limitare l'accessibilità di una variabile. La protezione viene analizzata in seguito.

A seconda del loro campo di azione, si distinguono in particolare tre categorie più importanti di variabili: variabili appartenenti alla classe (*member variable*), variabili locali e parametri dei metodi.

Variabili appartenenti alla classe

Queste variabili appartengono alle classi e come tali sono dichiarate all'interno delle classi stesse, ma all'esterno dei metodi. L'esempio seguente mostra la dichiarazione della variabile '**`serveAQualcosa`**' come parte della classe '**`FaQualcosa`**'.

```
class FaQualcosa
{
    int serveAQualcosa = 0;

    // Dichiarazione dei metodi
    ...
}
```

Variabili locali

Sono variabili dichiarate all'interno dei metodi. Vengono create alla chiamata del metodo e distrutte alla sua conclusione. Per questo sono visibili solo all'interno del metodo che le dichiara.

Nell'esempio visto in precedenza, quello che visualizza 10 «x», la variabile '**`contatore`**' veniva dichiarata all'interno del metodo '**`main`**'.

Parametri dei metodi

Le variabili indicate in concomitanza con la dichiarazione di un metodo (quelle che appaiono tra parentesi tonde), vengono create nel momento della chiamata del metodo stesso e distrutte alla sua conclusione. Queste variabili contengono la copia degli argomenti utilizzati per la chiamata; in questo senso si dice che le chiamate ai metodi avvengono per valore.

Operatori ed espressioni

Gli operatori sono qualcosa che esegue un qualche tipo di funzione, su uno o due operandi, restituendo un valore. Il valore restituito è di tipo diverso a seconda degli operandi utilizzati. Per esempio, la somma di due interi genera un risultato intero.

Gli operandi descritti nelle sezioni seguenti sono solo quelli più comuni e importanti. In particolare, sono stati omessi quelli necessari al trattamento delle variabili in modo binario.

Operatori aritmetici

Gli operatori che intervengono su valori numerici sono elencati nella tabella u123.8.

Tabella u123.8. Elenco degli operatori aritmetici e di quelli di assegnamento relativi a valori numerici.

Operatore e operandi	Descrizione
`++`***op***	Incrementa di un'unità l'operando prima che venga restituito il suo valore.
op`++`	Incrementa di un'unità l'operando dopo averne restituito il suo valore.
`--`***op***	Decrementa di un'unità l'operando prima che venga restituito il suo valore.
op`--`	Decrementa di un'unità l'operando dopo averne restituito il suo valore.
`+`***op***	Non ha alcun effetto.
`-`***op***	Inverte il segno dell'operando.
op1 `+` ***op2***	Somma i due operandi.
op1 `-` ***op2***	Sottrae dal primo il secondo operando.
op1 `*` ***op2***	Moltiplica i due operandi.
op1 `/` ***op2***	Divide il primo operando per il secondo.
op1 `%` ***op2***	Modulo -- il resto della divisione tra il primo e il secondo operando.
var `=` ***valore***	Assegna alla variabile il valore alla destra.
op1 `+=` ***op2***	***op1*** `=` ***op1*** `+` ***op2***
op1 `-=` ***op2***	***op1*** `=` ***op1*** `-` ***op2***
op1 `*=` ***op2***	***op1*** `=` ***op1*** `*` ***op2***
op1 `/=` ***op2***	***op1*** `=` ***op1*** `/` ***op2***
op1 `%=` ***op2***	***op1*** `=` ***op1*** `%` ***op2***

Operatori di confronto e operatori logici

Gli operatori di confronto determinano la relazione tra due operandi. Il risultato dell'espressione composta da due operandi posti a confronto è di tipo booleano, rappresentabile in Java dalle costanti letterali '**`true`**' e '**`false`**'. Gli operatori di confronto sono elencati nella tabella u123.9.

Tabella u123.9. Elenco degli operatori di confronto. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 `==` ***op2***	*Vero* se gli operandi si equivalgono.
op1 `!=` ***op2***	*Vero* se gli operandi sono differenti.
op1 `<` ***op2***	*Vero* se il primo operando è minore del secondo.
op1 `>` ***op2***	*Vero* se il primo operando è maggiore del secondo.
op1 `<=` ***op2***	*Vero* se il primo operando è minore o uguale al secondo.
op1 `>=` ***op2***	*Vero* se il primo operando è maggiore o uguale al secondo.

Quando si vogliono combinare assieme diverse espressioni logiche, comprendendo in queste anche delle variabili che contengono un valore booleano, si utilizzano gli operatori logici (noti normalmente come: AND, OR, NOT, ecc.). Il risultato di un'espressione logica complessa è quello dell'ultima espressione elementare che sia stata valutata effettivamente. Gli operatori logici sono elencati nella tabella u123.10.

Tabella u123.10. Elenco degli operatori logici. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
`!` ***op***	Inverte il risultato logico dell'operando.
op1 `&&` ***op2***	Se il risultato del primo operando è *Falso* non valuta il secondo.
op1 `||` ***op2***	Se il risultato del primo operando è *Vero* non valuta il secondo.

Concatenamento di stringhe

Si è accennato al fatto che in Java, le stringhe siano oggetti e non tipi di dati primitivi. Esiste tuttavia la possibilità di indicare stringhe letterali nel modo consueto, attraverso la delimitazione con gli apici doppi.

Diverse stringhe possono essere concatenate, in modo da formare una stringa unica, attraverso l'operatore '**`+`**'.

```
public static void main (String[] args)
{
    int contatore = 0;

    while (contatore < 10)
      {
        contatore++;
        System.out.println ("Ciclo n. " + contatore);
      }
}
```

Nel pezzo di codice appena mostrato, appare in particolare l'istruzione seguente:

```
        System.out.println ("Ciclo n. " + contatore);
```

L'espressione '**`"Ciclo n. " + contatore`**' si traduce nel risultato seguente:

```
Ciclo n. 1
Ciclo n. 2
…
Ciclo n. 10
```

In pratica, il contenuto della variabile '**`contatore`**' viene convertito automaticamente in stringa e unito alla costante letterale precedente.

Strutture di controllo del flusso

Le strutture di controllo del flusso delle istruzioni sono molto simili a quelle del linguaggio C. In particolare, dove può essere messa un'istruzione si può mettere anche un gruppo di istruzioni delimitate dalle parentesi graffe.

Normalmente, le strutture di controllo del flusso basano questo controllo sulla verifica di una condizione espressa all'interno di parentesi tonde.

Struttura condizionale: «if»

```
if (condizione) istruzione
```

```
if (condizione) istruzione else istruzione
```

Se la condizione si verifica, viene eseguita l'istruzione (o il gruppo di istruzioni) seguente; quindi il controllo passa alle istruzioni successive alla struttura. Se viene utilizzato '**`else`**', nel caso non si verifichi la condizione, viene eseguita l'istruzione che ne segue. Vengono mostrati alcuni esempi.

```
int importo;
…
if (importo > 10000000) System.out.println ("L'offerta è vantaggiosa");
```

```
int importo;
int memorizza;
…
if (importo > 10000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è vantaggiosa");
  }
else
  {
    System.out.println ("Lascia perdere");
  }
```

```
int importo;
int memorizza;
…
if (importo > 10000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è vantaggiosa");
  }
else if (importo > 5000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è accettabile");
  }
else
  {
    System.out.println ("Lascia perdere");
  }
```

Struttura di selezione: «switch»

L'istruzione '**`switch`**' è un po' troppo complessa per essere rappresentata in modo chiaro attraverso uno schema sintattico. In generale, l'istruzione '**`switch`**' permette di **saltare** a una certa posizione della struttura, in base al risultato di un'espressione. L'esempio seguente mostra la visualizzazione del nome del mese, in base al valore di un intero:

```
        int mese;
        …
        switch (mese)
          {
            case 1: System.out.println ("gennaio"); break;
            case 2: System.out.println ("febbraio"); break;
            case 3: System.out.println ("marzo"); break;
            case 4: System.out.println ("aprile"); break;
            case 5: System.out.println ("maggio"); break;
            case 6: System.out.println ("giugno"); break;
            case 7: System.out.println ("luglio"); break;
            case 8: System.out.println ("agosto"); break;
            case 9: System.out.println ("settembre"); break;
            case 10: System.out.println ("ottobre"); break;
            case 11: System.out.println ("novembre"); break;
            case 12: System.out.println ("dicembre"); break;
          }
```

Come si vede, dopo l'istruzione con cui si emette il nome del mese attraverso lo standard output, viene aggiunta un'istruzione di salto '**`break`**', che serve a uscire dalla struttura, perché altrimenti le istruzioni del caso successivo, se c'è, verrebbero eseguite. Infatti, un gruppo di casi può essere raggruppato assieme, quando si vuole che questi eseguano lo stesso gruppo di istruzioni:

```
        int mese;
        int giorni;
        …
        switch (mese)
          {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                giorni = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                giorni = 30;
                break;
            case 2:
                if (((anno % 4 == 0) && !(anno % 100 = 0))
                    || (anno % 400 == 0))
                    giorni = 29;
                else
                    giorni = 28;
                break;
          }
```

È anche possibile definire un caso predefinito che si verifichi quando nessuno degli altri si avvera:

```
        int mese;
        …
        switch (mese)
          {
            case 1: System.out.println ("gennaio"); break;
            case 2: System.out.println ("febbraio"); break;
            …
            case 11: System.out.println ("novembre"); break;
            case 12: System.out.println ("dicembre"); break;
            default: System.out.println ("mese non corretto"); break;
          }
```

Iterazione con condizione di uscita iniziale: «while»

```
while (condizione) istruzione
```

'**`while`**' esegue un'istruzione, o un gruppo di queste, finché la condizione restituisce il valore *Vero*. La condizione viene valutata prima di eseguire il gruppo di istruzioni e poi ogni volta che termina un ciclo, prima dell'esecuzione del successivo. Segue il pezzo dell'esempio già visto, di quella classe che visualizza 10 volte la lettera «x»:

```
        int contatore = 0;

        while (contatore < 10)
          {
            contatore++;
            System.out.println ("x");
          }
```

Nel blocco di istruzioni di un ciclo '**`while`**', ne possono apparire alcune particolari:

- '**`break`**'
  esce definitivamente dal ciclo '**`while`**';
- '**`continue`**'
  interrompe l'esecuzione del gruppo di istruzioni e riprende dalla valutazione della condizione.

L'esempio seguente è una variante del ciclo di visualizzazione mostrato sopra, modificato in modo da vedere il funzionamento di '**`break`**'. Si osservi che '**`while (true)`**' equivale a un ciclo senza fine, perché la condizione è sempre vera:

```
        int contatore = 0;

        while (true)
          {
            if (contatore >= 10)
              {
                break;
              }
            contatore++;
            System.out.println ("x");
          }
```

Iterazione con condizione di uscita finale: «do-while»

```
do blocco_di_istruzioni while (condizione);
```

'**`do`**' esegue un gruppo di istruzioni una volta e poi ne ripete l'esecuzione finché la condizione restituisce il valore *Vero*.

Iterazione enumerativa: «for»

```
for (espressione1; espressione2; espressione3) istruzione
```

Questa è la forma tipica di un'istruzione '**`for`**', in cui la prima espressione corrisponde all'assegnamento iniziale di una variabile, la seconda a una condizione che deve verificarsi fino a che si vuole che sia eseguita l'istruzione (o il gruppo di istruzioni), mentre la terza serve per l'incremento o decremento della variabile inizializzata con la prima espressione. In pratica, potrebbe esprimersi nella sintassi seguente:

```
for (var = n; condizione; var++) istruzione
```

Il ciclo '**for**' potrebbe essere definito anche in maniera differente, più generale: la prima espressione viene eseguita una volta sola all'inizio del ciclo; la seconda viene valutata all'inizio di ogni ciclo e il gruppo di istruzioni viene eseguito solo se il risultato è *Vero*; l'ultima viene eseguita alla fine dell'esecuzione del gruppo di istruzioni, prima che si ricominci con l'analisi della condizione.

Il vecchio esempio banale, in cui veniva visualizzata per 10 volte una «x», potrebbe tradursi nel modo seguente, attraverso l'uso di un ciclo '**for**':

```
        int contatore;

        for (contatore = 0; contatore < 10; contatore++)
          {
            System.out.println ("x");
          }
```

Array e stringhe

In Java, array e stringhe sono oggetti. In pratica, la variabile che contiene un array o una stringa è in realtà un riferimento alla struttura di dati rispettiva.

Array

La dichiarazione di un array avviene in Java in modo molto semplice, senza l'indicazione esplicita del numero di elementi. La dichiarazione avviene come se si trattasse di un tipo di dati normale, con la differenza che si aggiungono una coppia di parentesi quadre a sottolineare che si tratta di un array di elementi di quel tipo. L'esempio seguente dichiara che '**arrayDiInteri**' è un array in cui gli elementi sono di tipo intero ('**int**'), senza specificare quanti siano:

```
int[] arrayDiInteri;
```

Per fare in modo che l'array esista effettivamente, occorre che questo sia inizializzato, fornendogli gli elementi. Si usa per questo l'operatore '**new**' seguito dal tipo di dati con il numero di elementi racchiuso tra parentesi quadre. L'esempio seguente assegna alla variabile '**arrayDiInteri**' il riferimento a un array composto da sette interi:

```
arrayDiInteri = new int[7];
```

Nella pratica, è normale inizializzare l'array quando lo si dichiara; per cui, quanto già visto si può ridurre all'esempio seguente:

```
int[] arrayDiInteri = new int[7];
```

Il riferimento a un elemento di un array avviene aggiungendo al nome della variabile che rappresenta l'array stesso, il numero dell'elemento, racchiuso tra parentesi quadre. Come nel linguaggio C, il primo elemento si raggiunge con l'indice zero, mentre l'ultimo corrisponde alla dimensione meno uno.

Si è detto che gli array sono oggetti. In particolare, è possibile determinare la dimensione di un array, espressa in numero di elementi, leggendo il contenuto della variabile '**length**' dell'oggetto array. Nel caso dell'esempio già visto, si tratta di leggere il contenuto di '**arrayDiInteri.length**'.

L'esempio seguente mostra una scansione di un array, indicando una condizione di interruzione del ciclo indipendente dalla conoscenza anticipata della dimensione dell'array stesso. In particolare, la variabile '**i**' viene dichiarata contestualmente con la sua inizializzazione, nella prima espressione di controllo del ciclo '**for**':

```
for (int i = 0; i < arrayDiInteri.length; i++) {
    arrayDiInteri[i] = i;
}
```

Un array può contenere sia elementi primitivi che riferimenti a oggetti. In questo modo si possono avere gli array multidimensionali. L'esempio seguente rappresenta il modo in cui può essere definito un array 3x2 di interi e anche come scandirne i vari elementi:

```
/**
 *      Matrice3x2App.java
 *      Esempio di uso di array multidimensionali.
 */

import java.lang.*; // predefinita

class Matrice3x2App
{
    public static void main (String[] args)
    {
        int[][] matrice = new int[3][2];

        for (int i = 0; i < matrice.length; i++)
          {
            for (int j = 0; j < matrice[i].length; j++)
              {
                matrice[i][j] = 1000 + j + i * 10;
                System.out.println ("matrice[" + i + "][" + j + "] = "
                                    + matrice[i][j]);
              }
          }
    }
}
```

L'esecuzione di questo piccolo programma, genera il risultato seguente:

```
matrice[0][0] = 1000
matrice[0][1] = 1001
matrice[1][0] = 1010
matrice[1][1] = 1011
matrice[2][0] = 1020
matrice[2][1] = 1021
```

Stringhe

Le stringhe in Java sono oggetti e se ne distinguono due tipi: stringhe costanti e stringhe variabili. La distinzione è utile perché questi due tipi di oggetti hanno bisogno di una forma di rappresentazione diversa. Così, ciò porta a un'ottimizzazione del programma, che per una stringa costante richiede meno risorse rispetto a una stringa che deve essere variabile, oltre a migliorare altri aspetti legati alla sicurezza.

La dichiarazione di una variabile che possa contente un riferimento a un oggetto stringa-costante, si ottiene con la dichiarazione seguente:

```
String variabile;
```

In pratica, si dichiara che la variabile può contenere un riferimento a un oggetto di tipo '**String**'. La creazione di questo oggetto '**String**' si ottiene come nel caso degli array, utilizzando l'operatore '**new**'.

```
new String (stringa);
```

L'esempio seguente crea la variabile '**stringaCostante**' di tipo '**String**' e la inizializza assegnandoci il riferimento a una stringa:

```
String stringaCostante = new String ("Ciao ciao.");
```

Fortunatamente, si possono utilizzare anche delle costanti letterali pure e semplici. Per cui la stringa '**"Ciao ciao."**' è già di per sé un oggetto stringa-costante.

Si è già accennato al fatto che le stringhe-costanti possono essere concatenate facilmente utilizzando l'operatore '**+**':

```
"Ciao " + "come " + "stai?"
```

L'esempio restituisce un'unica stringa-costante, come quella seguente:

```
"Ciao come stai?"
```

Inoltre, in questi concatenamenti, entro certi limiti, possono essere inseriti elementi diversi da stringhe, come nell'esempio seguente, dove il contenuto numerico intero della variabile '**contatore**' viene convertito automaticamente in stringa prima di essere emesso attraverso lo standard output.

```
int contatore = 0;

while (contatore < 10)
{
    contatore++;
    System.out.println ("Ciclo n. " + contatore);
}
```

Le stringhe variabili sono oggetti di tipo '**StringBuffer**' e vengono descritte più avanti.

Metodo «main()»

Si è accennato al fatto che una classe che non eredita esplicitamente da un'altra, richiede l'esistenza del metodo '**`main()`**' e viene detta applicazione. Questo metodo deve avere una forma precisa e si tratta di quello che viene chiamato automaticamente quando si avvia il binario Java corrispondente alla classe stessa. Senza questa convenzione, non ci sarebbe un modo per avviare un programma Java.

```
public static void main (String[] args) { istruzioni }
```

Nella sintassi indicata, le parentesi graffe fanno parte della dichiarazione del metodo e delimitano un gruppo di istruzioni.

args

È Importante osservare l'unico parametro del metodo '**`main()`**': l'array '**`args`**' composto da elementi di tipo '**`String`**'. Questo array contiene gli argomenti passati al programma Java attraverso la riga di comando.

L'esempio seguente, mostra come si può leggere il contenuto di questo array, tenendo presente che non si conosce inizialmente la sua dimensione. L'esempio emette separatamente, attraverso lo standard output, l'elenco degli argomenti ricevuti.

```
/**
 *  LeggiArgomentiApp.java
 *  Legge gli argomenti e gli emette attraverso lo standard output.
 */

import java.lang.*; // predefinita

class LeggiArgomentiApp
{
    public static void main (String[] args)
    {
        int i;

        for (i = 0; i < args.length; i++)
          {
            System.out.println (args[i]);
          }
    }
}
```


Java: programmazione a oggetti

Creazione e distruzione di un oggetto 1013
- Dichiarazione dell'oggetto 1013
- Instanza di un oggetto 1014
- Metodo costruttore 1014
- Utilizzo degli oggetti 1014
- Distruzione di un oggetto 1015

Classi 1015
- Variabili 1016
- Metodi 1016
- Specificatore di accesso 1017

Sottoclassi 1018
- super 1018
- this 1018

Interfacce 1019
- Contenuto di un'interfaccia 1019
- Utilizzo di un'interfaccia 1019

Pacchetti di classi 1020
- Collocazione dei pacchetti 1020
- Utilizzo di classi di un pacchetto 1021

Esempi 1022
- Oggetti e messaggi 1022
- Variabili di istanza e variabili statiche 1022
- Ereditarietà 1023
- Metodi di istanza e metodi statici 1023

Il capitolo precedente ha introdotto l'uso del linguaggio Java per arrivare a scrivere programmi elementari, utilizzando i metodi come se fossero delle funzioni pure e semplici. In questo capitolo si introducono gli oggetti secondo Java.

Creazione e distruzione di un oggetto

Un oggetto è un'***istanza*** di una classe, come una copia ottenuta da uno stampo. Così come nel caso della creazione di una variabile contenente un tipo di dati primitivo, si distinguono due fasi: la dichiarazione e l'inizializzazione. Trattandosi di un oggetto, l'inizializzazione richiede prima la creazione dell'oggetto stesso, in modo da poter assegnare alla variabile il riferimento di questo.

Dichiarazione dell'oggetto

La dichiarazione di un oggetto è precisamente la dichiarazione di una variabile atta a contenere un riferimento a un particolare tipo di oggetto, specificato dalla classe che può generarlo.

```
classe variabile
```

La sintassi appena mostrata dovrebbe essere sufficientemente chiara. Nell'esempio seguente si dichiara la variabile '**`miaStringa`**' predisposta a contenere un riferimento a un oggetto di tipo '**`String`**'.

```
String miaStringa;
```

La semplice dichiarazione della variabile non basta a creare l'oggetto, in quanto così si crea solo il contenitore adatto.

Instanza di un oggetto

L'istanza di un oggetto si ottiene utilizzando l'operatore '**new**' seguito da una chiamata a un metodo particolare il cui scopo è quello di inizializzare opportunamente il nuovo oggetto che viene creato. In pratica, '**new**' alloca memoria per il nuovo oggetto, mentre il metodo chiamato lo prepara. Alla fine, viene restituito un riferimento all'oggetto appena creato.

L'esempio seguente, definisce la variabile '**miaStringa**' predisposta a contenere un riferimento a un oggetto di tipo '**String**', creando contestualmente un nuovo oggetto '**String**' inizializzato in modo da contenere un messaggio di saluto.

```
String miaStringa = new String ("Ciao ciao.");
```

Metodo costruttore

L'inizializzazione di un oggetto viene svolta da un metodo specializzato per questo scopo: il ***costruttore***. Una classe può fornire diversi metodi costruttori che possono servire a inizializzare in modo diverso l'oggetto che si ottiene. Tuttavia, convenzionalmente, ogni classe fornisce sempre un metodo il cui nome corrisponde a quello della classe stessa, ed è senza argomenti. Questo metodo esiste anche se non viene indicato espressamente all'interno della classe.

Java consente di utilizzare lo stesso nome per metodi che accettano argomenti in quantità o tipi diversi, perché è in grado di distinguere il metodo chiamato effettivamente in base agli argomenti forniti. Questo meccanismo permette di avere classi con diversi metodi costruttori, che richiedono una serie differente di argomenti.

Utilizzo degli oggetti

Finché non si utilizza in pratica un oggetto non si può apprezzare, né comprendere, la programmazione a oggetti. Un oggetto è una sorta di scatola nera a cui si accede attraverso variabili e metodi dell'oggetto stesso.

Si indica una variabile o un metodo di un oggetto aggiungendo un punto ('.') al riferimento dell'oggetto, seguito dal nome della variabile o del metodo da raggiungere. Variabili e metodi si distinguono perché questi ultimi possono avere una serie di argomenti racchiusi tra parentesi (se non hanno argomenti, vengono usate le parentesi senza nulla all'interno).

riferimento_all'oggetto . variabile

riferimento_all'oggetto . metodo ( )

Prima di proseguire, è bene soffermarsi sul significato si tutto questo. Indicare una cosa come '**oggetto.variabile**', significa raggiungere una variabile appartenente a una particolare struttura di dati, che è appunto l'oggetto. In un certo senso, ciò si avvicina all'accesso a un elemento di un array.

Un po' più difficile è comprendere il senso di un metodo di un oggetto. Indicare '**oggetto.metodo()**' significa chiamare una funzione che interviene in un ambiente particolare: quello dell'oggetto.

A questo punto, è necessario chiarire che il riferimento all'oggetto è qualunque cosa in grado di restituire un riferimento a questo. Normalmente si tratta di una variabile, ma questa potrebbe appartenere a sua volta a un altro oggetto. È evidente che sta poi al programmatore cercare si scrivere un programma leggibile.

Nella programmazione a oggetti si insegna comunemente che si dovrebbe evitare di accedere direttamente alle variabili, cercando di utilizzare il più possibile i metodi. Si immagini l'esempio seguente che è solo ipotetico:


```
class Divisione
{
    public int x;
    public int y;
    public calcola ()
    {
        return x/y;
    }
}
```

Se venisse creato un oggetto a partire da questa classe, si potrebbe modificare il contenuto delle variabili e quindi richiamare il calcolo, come nell'esempio seguente:

```
Divisione div = new Divisione ();
div.x = 10;
div.y = 5;
System.out.println ("Il risultato è " + div.calcola ());
```

Però, se si tenta di dividere per zero si ottiene un errore irreversibile. Se invece esistesse un metodo che si occupa di ricevere i dati da inserire nelle variabili, verificando prima che siano validi, si potrebbe evitare di dover prevedere questi inconvenienti.

L'esempio mostrato è volutamente banale, ma gli oggetti (ovvero le classi che li generano) possono essere molto complessi; pertanto, la loro utilità sta proprio nel fatto di poter inserire al loro interno tutti i meccanismi di filtro e controllo necessari al loro buon funzionamento.

In conclusione, in Java è considerato un buon approccio di programmazione l'utilizzo delle variabili solo in lettura, senza poterle modificarle direttamente dall'esterno dell'oggetto.

La chiamata di un metodo di un oggetto viene anche detta ***messaggio***, per sottolineare il fatto che si invia un'informazione (eventualmente composta dagli argomenti del metodo) all'oggetto stesso.

Distruzione di un oggetto

In Java, un oggetto viene eliminato automaticamente quando non esistono più riferimenti alla sua struttura. In pratica, se viene creato un oggetto assegnando il suo riferimento a una variabile, quando questa viene eliminata perché è terminato il suo campo di azione, anche l'oggetto viene eliminato.

Tuttavia, l'eliminazione di un oggetto non può essere presa tanto alla leggera. Un oggetto potrebbe avere in carico la gestione di un file che deve essere chiuso prima dell'eliminazione dell'oggetto stesso. Per questo, esiste un sistema di eliminazione degli oggetti, definito *garbage collector*, o più semplicemente ***spazzino***, che prima di eliminare un oggetto gli permette di eseguire un metodo conclusivo: '**finalize()**'. Questo metodo potrebbe occuparsi di chiudere i file rimasti aperti e di concludere ogni altra cosa necessaria.

Classi

Le classi sono lo stampo, o il prototipo, da cui si ottengono gli oggetti. La sintassi per la creazione di una classe è la seguente. Le parentesi graffe fanno parte dell'istruzione necessaria a creare la classe e ne delimitano il contenuto, ovvero il corpo, costituito dalla dichiarazione di variabili e metodi. Convenzionalmente, il nome di una classe inizia con una lettera maiuscola.

[***modificatore***] `class` ***classe*** [`extends` ***classe_superiore***] [`implements` ***elenco_interfacce***] `{…}`

Il modificatore può essere costituito da uno dei nomi seguenti, a cui corrisponde un valore differente della classe.

Modificatore	Descrizione
`public`	Quando la classe è accessibile anche al di fuori del pacchetto di classi cui appartiene, si utilizza il modificatore '**public**'. Se questo non viene indicato, la classe è accessibile solo all'interno del pacchetto cui appartiene.
`abstract`	Quando una classe serve solo come modello astratto per generare altre sottoclassi si utilizza il modificatore '**abstract**'.
`final`	Quando si vuole evitare che una classe possa generare altre sottoclassi si indica il modificatore '**final**'.

Tutte le classi ereditano automaticamente dalla classe '**`java.lang.Object`**', quando non viene dichiarano espressamente di ereditare da un'altra. La dichiarazione esplicita di volere ereditare da una classe particolare, si ottiene attraverso la parola chiave '**`extends`**' seguita dal nome della classe stessa.

A fianco dell'eredità da un'altra classe, si abbina il concetto di interfaccia, che rappresenta solo un'impostazione a cui si vuole fare riferimento. Questa impostazione non è un'eredità, ma solo un modo per definire una struttura standard che si vuole sia attuata nella classe che si va a creare.

L'eredità avviene sempre solo da una classe, mentre le interfacce che si vogliono utilizzare nella classe possono essere diverse. Se si vogliono specificare più interfacce, i nomi di queste vanno separati con la virgola.

Nel corpo di una classe possono apparire dichiarazioni di variabili e metodi, definiti anche ***membri*** della classe.

Variabili

Le variabili dichiarate all'interno di una classe, ma all'esterno dei metodi, fanno parte dei cosiddetti membri, sottintendendo con questo che si tratta di componenti delle classi (anche i metodi sono definiti membri). La dichiarazione di una variabile di questo tipo, può essere espressa in forma piuttosto articolata. La sintassi seguente mostra solo gli aspetti più importanti.

[*specificatore_di_accesso*] [`static`] [`final`] *tipo variabile* [= *valore_iniziale*]

Lo specificatore di accesso rappresenta la visibilità della variabile ed è qualcosa di diverso dal campo di azione, che al contrario rappresenta il ciclo vitale di questa. Per definire questa visibilità si utilizza una parola chiave il cui elenco e significato è descritto nella sezione i124.2.3.

La parola chiave '**`static`**' indica che si tratta di una variabile appartenente strettamente alla classe, mentre la mancanza di questa indicazione farebbe sì che si tratti di una variabile di istanza. Quando si dichiarano variabili statiche, si intende che ogni istanza (ogni oggetto generato) della classe che le contiene faccia riferimento alle stesse variabili. Al contrario, in presenza di variabili non statiche, ogni istanza della classe genera una nuova copia indipendente di queste variabili.

La parola chiave '**`final`**' indica che si tratta di una variabile che non può essere modificata, in pratica si tratta di una costante. In tal caso, la variabile deve essere inizializzata contemporaneamente alla sua creazione.

Il nome di una variabile inizia convenzionalmente con una lettera minuscola, ma quando si tratta di una costante, si preferisce usare solo lettere maiuscole.

Metodi

I metodi, assieme alle variabili dichiarate all'esterno dei metodi, fanno parte dei cosiddetti membri delle classi. La sintassi seguente mostra solo gli aspetti più importanti della dichiarazione di un metodo. Le parentesi graffe fanno parte dell'istruzione necessaria a creare il metodo e ne delimitano il contenuto, ovvero il corpo.

[*specificatore_di_accesso*] [`static`] [`abstract`] [`final`]
tipo_restituito ↩
↪*metodo*([*elenco_parametri*]) [`throws` *elenco_eccezioni*] {…}

Lo specificatore di accesso rappresenta la visibilità del metodo. Per definire questa visibilità si utilizza una parola chiave il cui elenco e significato è descritto nella sezione i124.2.3.

La parola chiave '**`static`**' indica che si tratta di un metodo appartenente strettamente alla classe, mentre la mancanza di questa indicazione farebbe sì che si tratti di un metodo di istanza. I metodi statici possono accedere solo a variabili statiche; di conseguenza, per essere chiamati non c'è bisogno di creare un'istanza della classe che li contiene. Il metodo normale, non statico, richiede la creazione di un'istanza della classe che lo contiene per poter essere eseguito.

La parola chiave '**`abstract`**' indica che si tratta della struttura di un metodo, del quale vengono indicate solo le caratteristiche esterne, senza definirne il contenuto.

La parola chiave '**`final`**' indica che si tratta di un metodo che non può essere dichiarato nuovamente, nel senso che non può essere modificato in una sottoclasse eventuale.

Il tipo di dati restituito viene indicato prima del nome, utilizzando la stessa definizione che si darebbe a una variabile normale. Nel caso si tratti di un metodo che non restituisce alcunché, si utilizza la parola chiave '**`void`**'.

Il nome di un metodo inizia convenzionalmente con una lettera minuscola, come nel caso delle variabili.

L'elenco di parametri è composto da nessuno o più nomi di variabili precedute dal tipo. Questa elencazione corrisponde implicitamente alla creazione di altrettante variabili locali contenenti il valore corrispondente (in base alla posizione) utilizzato nella chiamata.

La parola chiave '**`throws`**' introduce un elenco di oggetti utili per superare gli errori generati durante l'esecuzione del programma. Tale gestione non viene analizzata in questa documentazione su Java.

Sovraccarico

Java ammette il ***sovraccarico*** dei metodi. Questo significa che, all'interno della stessa classe, si possono dichiarare metodi differenti con lo stesso nome, purché sia diverso il numero o il tipo di parametri che possono accettare. In pratica, il metodo giusto viene riconosciuto alla chiamata in base agli argomenti che vengono forniti.

Chiamata di un metodo

La chiamata di un metodo avviene in modo simile a quanto si fa con le chiamate di funzione negli altri linguaggi. La differenza fondamentale sta nella necessità di indicare l'oggetto a cui si riferisce la chiamata.

Java consente anche di eseguire chiamate di metodi riferiti a una classe, quando si tratta di metodi statici.

Specificatore di accesso

Lo specificatore di accesso di variabili e metodi permette di limitare o estendere l'accessibilità di questi, sia per una questione di ordine (nascondendo i nomi di variabili e metodi cui non ha senso accedere da una posizione determinata), sia per motivi di sicurezza.

La tabella u124.6 mostra in modo sintetico e chiaro l'accessibilità dei componenti in base al tipo di specificatore indicato.

Tabella u124.6. Accessibilità di variabili e metodi in base all'uso di specificatori di accesso.

Specificatore	Classe	Sottoclasse	Pacchetto di classi	Altri
`package`	X		X	
`private`	X			
`protected`	X	X	X	
`public`	X	X	X	X

Se le variabili o i metodi vengono dichiarati senza l'indicazione esplicita di uno specificatore di accesso, viene utilizzato il tipo '**`package`**' in modo predefinito.

Sottoclassi

Una sottoclasse è una classe che eredita esplicitamente da un'altra. A questo proposito, è il caso di ripetere che tutte le classi ereditano in modo predefinito da '**`java.lang.Object`**', se non viene specificato diversamente attraverso la parola chiave '**`extends`**'.

Quando si crea una sottoclasse, si ereditano tutte le variabili e i metodi che compongono la classe, salvo quei componenti che risultano oscurati dallo specificatore di accesso. Tuttavia, la classe può dichiarare nuovamente alcuni di quei componenti e si può ancora accedere a quelli della classe precedente, nonostante tutto.

super

La parola chiave '**`super`**' rappresenta un oggetto contenente esclusivamente componenti provenienti dalla classe di livello gerarchico precedente. Questo permette di accedere a variabili e metodi che la classe dell'oggetto in questione ha ridefinito. L'esempio seguente mostra la dichiarazione di due classi: la seconda estende la prima.

```
class MiaClasse
{
    int intero;
    void mioMetodo ()
    {
        intero = 100;
    }
}
```

```
class MiaSottoclasse extends MiaClasse
{
    int intero;
    void mioMetodo ()
    {
        intero = 0;
        super.mioMetodo ();
        System.out.println (intero);
        System.out.println (super.intero);
    }
}
```

La coppia di classi mostrata sopra è fatta per generare un oggetto a partire dalla seconda, quindi per eseguire il metodo '**`mioMetodo()`**' su questo oggetto. Il metodo a essere eseguito effettivamente è quello della sottoclasse.

Quando ci si comporta in questo modo, ridefinendo un metodo in una sottoclasse, è normale che questo richiami il metodo della classe superiore, in modo da aggiungere solo il codice sorgente che serve in più. In questo caso, viene richiamato il metodo omonimo della classe superiore utilizzando '**`super`**' come riferimento.

Nello stesso modo, è possibile accedere alla variabile '**`intero`**' della classe superiore, anche se in quella attuale tale variabile viene ridefinita.

È il caso di osservare che la parola chiave '**`super`**' ha senso solo quando dalla classe si genera un oggetto. Quando si utilizzano metodi e variabili statici per evitare di dover generare l'istanza di un oggetto, non è possibile utilizzare questa tecnica per raggiungere metodi e variabili di una classe superiore.

this

La parola chiave '**`this`**' permette di fare riferimento esplicitamente all'oggetto stesso. Ciò può essere utile in alcune circostanze, come nell'esempio seguente:

```
class MiaClasse
{
    int imponibile;
    int imposta;
    void datiFiscali (int imponibile, int imposta)
    {
        this.imponibile = imponibile;
        this.imposta = imposta;
    }
    …
}
```

La classe appena mostrata dichiara due variabili che servono a conservare le informazioni su imponibile e imposta. Il metodo

'**`datiFiscali()`**' permette di modificare questi dati in base agli argomenti con cui viene chiamato.

Per comodità, il metodo indica con gli stessi nomi le variabili utilizzate per ricevere i valori delle chiamate. Tali variabili diventano locali e oscurano le variabili di istanza omonime. Per poter accedere alle variabili di istanza si utilizza quindi la parola chiave '**`this`**'.

Anche in questa situazione, la parola chiave '**`this`**' ha senso solo quando dalla classe si genera un oggetto.

Interfacce

In Java, l'interfaccia è una raccolta di costanti e di definizioni di metodi senza attuazione. In un certo senso, si tratta di una sorta di prototipo di classe. Le interfacce non seguono la gerarchia delle classi perché rappresentano una struttura indipendente: un'interfaccia può ereditare da una o più interfacce definite precedentemente (al contrario delle classi che possono ereditare da una sola classe superiore), ma non può ereditare da una classe.

Nel caso di interfacce, non è corretto parlare di ereditarietà, ma questo concetto rende l'idea di ciò che succede effettivamente.

La sintassi per la definizione di un'interfaccia, è la seguente:

[`public`] `interface` ***interfaccia*** [`extends` ***elenco_interfacce_superiori***] {…}

Il modificatore '**`public`**' fa in modo che l'interfaccia sia accessibile a qualunque classe, indipendentemente dal pacchetto di classi cui questa possa appartenere. Al contrario, se non viene utilizzato, l'interfaccia risulta accessibile solo alle classi dello stesso pacchetto.

La parola chiave '**`extends`**' permette di indicare una o più interfacce superiori da cui ereditare.

Contenuto di un'interfaccia

Un'interfaccia può contenere solo la dichiarazione di costanti e di metodi astratti (senza attuazione). In pratica, non viene indicato alcuno specificatore di accesso e nessun'altra definizione che non sia il tipo, come nell'esempio seguente:

```
interface Raccoltina
{
    int LIMITEMASSIMO = 1000;

    void aggiungi (Object, obj);
    int conteggio ();
    …
}
```

Si intende implicitamente che le variabili siano '**`public`**', '**`static`**' e '**`final`**', inoltre si intende che i metodi siano '**`public`**' e '**`abstract`**'.

Come si può osservare dall'esempio, la definizione dei metodi termina con l'indicazione dei parametri. Il corpo dei metodi, ovvero la loro attuazione, non viene indicato, perché non è questo il compito di un'interfaccia.

Utilizzo di un'interfaccia

Un'interfaccia viene utilizzata in pratica quando una classe dichiara di attuare (realizzare) una o più interfacce. L'esempio seguente mostra l'utilizzo della parola chiave '**`implements`**' per dichiarare il legame con l'interfaccia vista nella sezione precedente:

```
class MiaClasse implements Raccoltina
{
    …
    void aggiungi (Object, obj)
    {
        …
    }
    int conteggio ()
    {
        …
    }
    …
}
```

In pratica, la classe che attua un'interfaccia, è obbligata a definire i metodi che l'interfaccia si limita a dichiarare in modo astratto. Si tratta quindi solo di una forma di standardizzazione e di controllo attraverso la stessa compilazione.

Pacchetti di classi

In Java si realizzano delle librerie di classi e interfacce attraverso la costruzione di pacchetti, come già accennato in precedenza. L'esempio seguente mostra due sorgenti Java, '**`Uno.java`**' e '**`Due.java`**' rispettivamente, appartenenti allo stesso pacchetto denominato '**`PaccoDono`**'. La dichiarazione dell'appartenenza al pacchetto viene fatta all'inizio, con l'istruzione '**`package`**'.

```
/**
 *  Uno.java
 *  Classe pubblica appartenente al pacchetto «PaccoDono».
 */

package PaccoDono;

public class Uno
{
    public void Visualizza ()
    {
        System.out.println ("Ciao Mondo - Uno");
    }
}
```

```
/**
 *  Due.java
 *  Classe pubblica appartenente al pacchetto «PaccoDono».
 */

package PaccoDono;

public class Due
{
    public void Visualizza ()
    {
        System.out.println ("Ciao Mondo - Due");
    }
}
```

Collocazione dei pacchetti

Quando si dichiara in un sorgente che una classe appartiene a un certo pacchetto, si intende che il binario Java corrispondente (il file '`.class`') sia collocato in una directory con il nome di quel pacchetto. Nell'esempio visto in precedenza si utilizzava la dichiarazione seguente:

```
package PaccoDono;
```

In tal modo, la classe (o le classi) di quel sorgente deve poi essere collocata nella directory '`PaccoDono/`'. Questa directory, a sua volta, deve trovarsi all'interno dei percorsi definiti nella variabile di ambiente '**`CLASSPATH`**'.

La variabile '**`CLASSPATH`**' è già stata vista in riferimento al file '`classes.zip`' o '`Klasses.jar`' (a seconda del tipo di compilatore e interprete Java), che si è detto contenere le librerie standard di Java. Tali librerie sono in effetti dei pacchetti di classi.

Il file '`classes.zip`' (o il file '`Klasses.jar`') potrebbe essere decompresso a partire dalla posizione in cui si trova, ma generalmente questo non si fa.

Se per ipotesi si decidesse di collocare la directory '`PaccoDono/`' a partire dalla propria directory personale, si potrebbe aggiungere nello script di configurazione della propria shell, qualcosa come l'istruzione seguente (adatta a una shell derivata da quella di Bourne).

```
CLASSPATH="$HOME:$CLASSPATH"
export CLASSPATH
```

Generalmente, per permettere l'accesso a pacchetti installati a partire dalla stessa directory di lavoro (nel caso del nostro esempio si tratterebbe di '`./PaccoDono/`'), si può aggiungere anche questa ai percorsi di '**`CLASSPATH`**'.

```
CLASSPATH=".:$HOME:$CLASSPATH"
export CLASSPATH
```

Utilizzo di classi di un pacchetto

L'utilizzo di classi da un pacchetto è già stato visto nei primi esempi, dove si faceva riferimento al fatto che ogni classe importa implicitamente le classi del pacchetto '**`java.lang`**'. Si importa una classe con un'istruzione simile all'esempio seguente:

```
import MioPacchetto.MiaClasse;
```

Per importare tutte le classi di un pacchetto, si utilizza un'istruzione simile all'esempio seguente:

```
import MioPacchetto.*;
```

In realtà, la dichiarazione dell'importazione di una o più classi, non è indispensabile, perché si potrebbe fare riferimento a quelle classi utilizzando un nome che comprende anche il pacchetto, separato attraverso un punto.

L'esempio seguente rappresenta un programma banale che utilizza le due classi mostrate negli esempi all'inizio di queste sezioni dedicate ai pacchetti:

```
/**
 *  MiaProva.java
 *  Classe che accede alle classi del pacchetto «PaccoDono».
 */

import PaccoDono.*;

class MiaProva
{
    public static void main (String[] args)
    {
        // Dichiara due oggetti dalle classi del pacchetto PaccoDono.
        Uno primo = new Uno ();
        Due secondo = new Due ();

        // Utilizza i metodi degli oggetti.
        primo.Visualizza ();
        secondo.Visualizza ();
    }
}
```

L'effetto che si ottiene è la sola emissione dei messaggi seguenti attraverso lo standard output:

```
Ciao Mondo - Uno
Ciao Mondo - Due
```

Se nel file non fosse stato dichiarato esplicitamente l'utilizzo di tutte le classi del pacchetto, sarebbe stato possibile accedere ugualmente alle sue classi utilizzando una notazione completa, che comprende anche il nome del pacchetto stesso. In pratica, l'esempio si modificherebbe come segue:

```
/**
 *  MiaProva.java
 *  Classe che accede alle classi del pacchetto «PaccoDono».
 */

class MiaProva
{
    public static void main (String[] args)
    {
        // Dichiara due oggetti dalle classi del pacchetto PaccoDono.
        PaccoDono.Uno primo = new PaccoDono.Uno ();
        PaccoDono.Due secondo = new PaccoDono.Due ();

        // Utilizza i metodi degli oggetti.
        primo.Visualizza ();
        secondo.Visualizza ();
    }
}
```

Esempi

Gli esempi mostrati nelle sezioni seguenti sono molto semplici, nel senso che si limitano a mostrare messaggi attraverso lo standard output. Si tratta quindi di pretesti per vedere come utilizzare quanto spiegato in questo capitolo. Viene usata in particolare la classe seguente per ottenere degli oggetti e delle sottoclassi:

```
/**
 *      SuperApp.java
 */

class SuperApp
{
    static int variabileStatica = 0; // variabile statica o di classe
    int variabileDiIstanza = 0;  // variabile di istanza

    // Nelle applicazioni è obbligatoria la presenza di questo metodo.
    public static void main (String[] args)
    {
        // Se viene avviata questa classe da sola, viene visualizzato
        // il messaggio seguente.
        System.out.println ("Ciao!");
    }

    // Metodo statico. Può essere usato per accedere solo alla
    // variabile statica.
    public static void metodoStatico ()
    {
        variabileStatica++;
        System.out.println
          ("La variabile statica ha raggiunto il valore "
           + variabileStatica);
    }

    // Metodo di istanza. Può essere usato per accedere sia alla
    // variabile statica che a quella di istanza.
    public void metodoDiIstanza ()
    {
        variabileStatica++;
        variabileDiIstanza++;
        System.out.println
          ("La variabile statica ha raggiunto il valore "
           + variabileStatica);
        System.out.println
          ("La variabile di istanza ha raggiunto il valore "
           + variabileDiIstanza);
    }
}
```

Oggetti e messaggi

Si crea un oggetto a partire da una classe, contenuta generalmente in un pacchetto. Nella sezione precedente è stata presentata una classe che si intende non appartenga ad alcun pacchetto di classi. Ugualmente può essere utilizzata per creare degli oggetti.

L'esempio seguente crea un oggetto a partire da quella classe e quindi esegue la chiamata del metodo '**`metodoDiIstanza`**', che emette due messaggi, per ora senza significato:

```
/**
 *      EsempioOggetti1App.java
 */

class EsempioOggetti1App
{
    public static void main (String[] args)
    {
        SuperApp oSuperApp = new SuperApp ();
        oSuperApp.metodoDiIstanza ();
    }
}
```

Variabili di istanza e variabili statiche

Le variabili di istanza appartengono all'oggetto, per cui, ogni volta che si crea un oggetto a partire da una classe si crea una nuova copia di queste variabili. Le variabili statiche, al contrario, appartengono a tutti gli oggetti della classe, per cui, quando si crea un nuovo oggetto, per queste variabili viene creato un riferimento all'unica copia esistente.

L'esempio seguente è una variante di quello precedente in cui si creano due oggetti dalla stessa classe, quindi viene chiamato lo stesso metodo, prima da un oggetto, poi dall'altro. Il metodo '**`metodoDiIstanza()`**' incrementa due variabili: una di istanza e l'altra statica.


```
/**
 *      EsempioOggetti2App.java
 */

class EsempioOggetti2App
{
    public static void main (String[] args)
    {
        SuperApp oSuperApp = new SuperApp ();
        SuperApp oSuperAppBis = new SuperApp ();

        oSuperApp.metodoDiIstanza ();
        oSuperAppBis.metodoDiIstanza ();
    }
}
```

Avviando l'eseguibile Java che deriva da questa classe, si ottiene la visualizzazione del testo seguente:

```
La variabile statica ha raggiunto il valore 1
La variabile di istanza ha raggiunto il valore 1
La variabile statica ha raggiunto il valore 2
La variabile di istanza ha raggiunto il valore 1
```

Le prime due righe sono generate dalla chiamata '**`oSuperApp.metodoDiIstanza()`**', mentre le ultime due da '**`oSuperAppBis.metodoDiIstanza()`**'. Si può osservare che l'incremento della variabile statica avvenuto nella prima chiamata riferita all'oggetto '**`oSuperApp`**' si riflette anche nel secondo oggetto, '**`oSuperAppBis`**', che mostra un valore più grande rispetto alla variabile di istanza corrispondente.

Ereditarietà

Nella programmazione a oggetti, il modo più naturale di acquisire variabili e metodi è quello di ereditare da una classe superiore che fornisca ciò che serve. L'esempio seguente mostra una classe che estende quella dell'esempio introduttivo ('**`SuperApp`**'), aggiungendo due metodi:

```
/**
 *      SottoclasseApp.java
 */

class SottoclasseApp extends SuperApp
{
    public static void decrementaStatico ()
    {
        variabileStatica--;
        System.out.println
          ("La variabile statica ha raggiunto il valore "
           + variabileStatica);
    }

    public void decrementaDiIstanza ()
    {
        variabileStatica--;
        variabileDiIstanza--;
        System.out.println
          ("La variabile statica ha raggiunto il valore "
           + variabileStatica);
        System.out.println
          ("La variabile di istanza ha raggiunto il valore "
           + variabileDiIstanza);
    }
}
```

Se dopo la compilazione si esegue questa classe, si ottiene l'esecuzione del metodo '**`main()`**' che è stato definito nella classe superiore. In pratica, si ottiene la visualizzazione di un semplice messaggio di saluto e nulla altro.

Metodi di istanza e metodi statici

Il metodo di istanza può accedere sia a variabili di istanza, sia a variabili statiche. Questo è stato visto nell'esempio del sorgente '`EsempioOggetti2App.java`', in cui il metodo '**`metodoDiIstanza()`**' incrementava e visualizzava il contenuto di due variabili, una di istanza e una statica.

I metodi statici possono accedere solo a variabili statiche, che come tali possono essere chiamati anche senza la necessità di creare un oggetto: basta fare riferimento direttamente alla classe. L'esempio mostra in che modo si possa chiamare il metodo '**`metodoStatico()`**' della classe '**`SuperApp`**', senza fare riferimento a un oggetto:

```
/**
 *      EsempioOggetti3App.java
 */

class EsempioOggetti3App
{
    public static void main (String[] args)
    {
        SuperApp.metodoStatico ();
    }
}
```

Nello stesso modo, quando in una classe si vuole chiamare un metodo senza dovere prima creare un oggetto, è necessario che i metodi in questione siano statici.

Java: esempi di programmazione

Problemi elementari di programmazione 1025
Somma tra due numeri positivi 1025
Moltiplicazione di due numeri positivi attraverso la somma 1026
Divisione intera tra due numeri positivi 1027
Elevamento a potenza 1027
Radice quadrata 1028
Fattoriale 1028
Massimo comune divisore 1029
Numero primo 1029
Scansione di array 1030
Ricerca sequenziale 1030
Ricerca binaria 1031
Algoritmi tradizionali 1032
Bubblesort 1032
Torre di Hanoi 1033
Quicksort 1033
Permutazioni 1035

Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Lo scopo di questi esempi è solo didattico, utilizzando forme non ottimizzate per la velocità di esecuzione.

Problemi elementari di programmazione 1025
Somma tra due numeri positivi 1025
Moltiplicazione di due numeri positivi attraverso la somma 1026
Divisione intera tra due numeri positivi 1027
Elevamento a potenza 1027
Radice quadrata 1028
Fattoriale 1028
Massimo comune divisore 1029
Numero primo 1029
Scansione di array 1030
Ricerca sequenziale 1030
Ricerca binaria 1031
Algoritmi tradizionali 1032
Bubblesort 1032
Torre di Hanoi 1033
Quicksort 1033
Permutazioni 1035

Problemi elementari di programmazione

In questa sezione vengono mostrati alcuni algoritmi elementari portati in Java.

Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
//
// java SommaApp <x> <y>
// Somma esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class SommaApp
{
    //
    static int somma (int x, int y)
```

```
    {
        int i;
        int z = x;
        //
        for (i = 1; i <= y; i++)
          {
            z++;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "+" + y + "=" + somma (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int somma (int x, int y)
    {
        int z = x;
        int i = 1;
        //
        while (i <= y)
          {
            z++;
            i++;
          }
        return z;
    }
```

Moltiplicazione di due numeri positivi attraverso la somma

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
//
// java MoltiplicaApp <x> <y>
// Moltiplica esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class MoltiplicaApp
{
    //
    static int moltiplica (int x, int y)
    {
        int i;
        int z = 0;
        //
        for (i = 1; i <= y; i++)
          {
            z = z + x;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "*" + y + "=" + moltiplica (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int moltiplica (int x, int y)
    {
        int z = 0;
        int i = 1;
        //
        while (i <= y)
          {
            z = z + x;
            i++;
          }
        return z;
    }
```


Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
//
// java DividiApp <x> <y>
// Divide esclusivamente valori positivi.
//
import java.lang.*; // predefinita
//
class DividiApp
{
    //
    static int dividi (int x, int y)
    {
        int z = 0;
        int i = x;
        //
        while (i >= y)
          {
            i = i - y;
            z++;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + ":" + y + "=" + dividi (x, y));
    }
}
//
```

Elevamento a potenza

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
//
// java ExpApp <x> <y>
// Elevamento a potenza di valori positivi interi.
//
import java.lang.*; // predefinita
//
class ExpApp
{
    //
    static int exp (int x, int y)
    {
        int z = 1;
        int i;
        //
        for (i = 1; i <= y; i++)
          {
            z = z * x;
          }
        return z;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println (x + "**" + y + "=" + exp (x, y));
    }
}
//
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**':

```
    static int exp (int x, int y)
    {
        int z = 1;
        int i = 1;
        //
        while (i <= y)
          {
            z = z * x;
            i++;
          }
        return z;
    }
```

Infine, si può usare anche un algoritmo ricorsivo:

```
    static int exp (int x, int y)
    {
        if (x == 0)
          {
            return 0;
          }
        else if (y == 0)
          {
            return 1;
          }
        else
          {
            return (x * exp (x, y-1));
          }
    }
```

Radice quadrata

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
//
// java RadiceApp <x>
// Estrazione della parte intera della radice quadrata.
//
import java.lang.*; // predefinita
//
class RadiceApp
{
    //
    static int radice (int x)
    {
        int z = 0;
        int t = 0;
        //
        while (true)
          {
            t = z * z;

            if (t > x)
              {
                //
                // È stato superato il valore massimo.
                //
                z--;
                return z;
              }
            z++;
          }
        //
        // Teoricamente, non dovrebbe mai arrivare qui.
        //
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        System.out.println ("radq(" + x + ")=" + radice (x));
    }
}
//
```

Fattoriale

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
//
// java FattorialeApp <x>
// Calcola il fattoriale di un valore intero.
//
import java.lang.*; // predefinita
//
class FattorialeApp
{
    //
    static int fattoriale (int x)
    {
        int i = x - 1;
        //
        while (i > 0)
          {
            x = x * i;
            i--;
          }
        return x;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        System.out.println (x + "! = " + fattoriale (x));
    }
}
//
```


In alternativa, l'algoritmo si può tradurre in modo ricorsivo:

```
    static int fattoriale (int x)
    {
        if (x > 1)
          {
            return (x * fattoriale (x - 1));
          }
        else
          {
            return 1;
          }
        //
        // Teoricamente non dovrebbe arrivare qui.
        //
    }
```

Massimo comune divisore

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione 62.3.7.

```
//
// java MCDApp <x> <y>
// Determina il massimo comune divisore tra due numeri interi positivi.
//
import java.lang.*; // predefinita
//
class MCDApp
{
    //
    static int mcd (int x, int y)
    {
        int i;
        int z = 0;
        //
        while (x != y)
          {
            if (x > y)
              {
                x = x - y;
              }
            else
              {
                y = y - x;
              }
          }
        return x;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        int y;
        //
        x = Integer.valueOf(args[0]).intValue ();
        y = Integer.valueOf(args[1]).intValue ();
        //
        System.out.println ("Il massimo comune divisore tra " + x
                            + " e " + y + " è " + mcd (x, y));
    }
}
//
```

Numero primo

Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione 62.3.8.

```
//
// java PrimoApp <x>
// Determina se un numero sia primo o meno.
//
import java.lang.*; // predefinita
//
class PrimoApp
{
    //
    static boolean primo (int x)
    {
        boolean primo = true;
        int i = 2;
        int j;
        //
        while ((i < x) && primo)
          {
            j = x / i;
            j = x - (j * i);
            //
            if (j == 0)
              {
                primo = false;
              }
            else
              {
                i++;
              }
          }
        return primo;
    }
```

```
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int x;
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        if (primo (x))
          {
            System.out.println (x + " è un numero primo");
          }
        else
          {
            System.out.println (x + " non è un numero primo");
          }
    }
}
//
```

Scansione di array

In questa sezione vengono mostrati alcuni algoritmi, legati alla scansione degli array, portati in Java.

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
//
// java RicercaSeqApp
//
import java.lang.*; // predefinita
//
class RicercaSeqApp
{
    //
    static int ricercaseq (int[] lista, int x, int a, int z)
    {
        int i;
        //
        // Scandisce l'array alla ricerca dell'elemento.
        //
        for (i = a; i <= z; i++)
          {
            if (x == lista[i])
              {
                return i;
              }
          }
        //
        // La corrispondenza non è stata trovata.
        //
        return -1;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length-1];
        int x;
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        for (i = 1; i < args.length; i++)
          {
            lista[i-1] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue la ricerca.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        i = ricercaseq (lista, x, 0, lista.length-1);
        //
        // Visualizza il risultato.
        //
        System.out.println (x + " si trova nella posizione "
                            + i + ".");
    }
}
//
```

Esiste anche una soluzione ricorsiva che viene mostrata nella subroutine seguente:


```
    static int ricercaseq (int[] lista, int x, int a, int z)
    {
        if (a > z)
          {
            //
            // La corrispondenza non è stata trovata.
            //
            return -1;
          }
        else if (x == lista[a])
          {
            return a;
          }
        else
          {
            return ricercaseq (lista, x, a+1, z);
          }
    }
```

Ricerca binaria

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione 62.4.2.

```
//
// java RicercaBinApp.java
//
import java.lang.*; // predefinita
//
class RicercaBinApp
{
    //
    static int ricercabin (int[] lista, int x, int a, int z)
    {
        int m;
        //
        // Determina l'elemento centrale.
        //
        m = (a + z) / 2;
        //
        if (m < a)
          {
            //
            // Non restano elementi da controllare: l'elemento cercato
            // non c'è.
            //
            return -1;
          }
        else if (x < lista[m])
          {
            //
            // Si ripete la ricerca nella parte inferiore.
            //
            return ricercabin (lista, x, a, m-1);
          }
        else if (x > lista[m])
          {
            //
            // Si ripete la ricerca nella parte superiore.
            //
            return ricercabin (lista, x, m+1, z);
          }
        else
          {
            //
            // m rappresenta l'indice dell'elemento cercato.
            //
            return m;
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length-1];
        int x;
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        x = Integer.valueOf(args[0]).intValue ();
        //
        for (i = 1; i < args.length; i++)
          {
            lista[i-1] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue la ricerca.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        i = ricercabin (lista, x, 0, lista.length-1);
        //
        // Visualizza il risultato.
        //
        System.out.println (x + " si trova nella posizione "
                            + i + ".");
    }
}
//
```

Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in Java.

Bubblesort

Il problema del Bubblesort è stato descritto nella sezione 62.5.1. Viene mostrata prima una soluzione iterativa e successivamente il metodo '**bsort**' in versione ricorsiva.

```
//
// java BSortApp
//
import java.lang.*; // predefinita
//
class BSortApp
{
    //
    static int[] bsort (int[] lista, int a, int z)
    {
        int scambio;
        int j;
        int k;
        //
        // Inizia il ciclo di scansione dell'array.
        //
        for (j = a; j < z; j++)
          {
            //
            // Scansione interna dell'array per collocare nella
            // posizione j l'elemento giusto.
            //
            for (k = j+1; k <= z; k++)
              {
                if (lista[k] < lista[j])
                  {
                    //
                    // Scambia i valori.
                    //
                    scambio = lista[k];
                    lista[k] = lista[j];
                    lista[j] = scambio;
                  }
              }
          }
        //
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento. Qui si restituisce ugualmente un
        // riferimento all'array ordinato.
        //
        return lista;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Ordina l'array.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        bsort (lista, 0, args.length-1);
        //
        // Visualizza il risultato.
        //
        for (i = 0; i < lista.length; i++)
          {
            System.out.println ("lista[" + i + "] = "
                                + lista[i]);
          }
    }

}
//
```

Segue il metodo '**bsort**' in versione ricorsiva:


```
    static int[] bsort (int[] lista, int a, int z)
    {
        int scambio;
        int k;
        //
        if (a < z)
          {
            //
            // Scansione interna dell'array per collocare nella
            // posizione a l'elemento giusto.
            //
            for (k = a+1; k <= z; k++)
              {
                if (lista[k] < lista[a])
                  {
                    //
                    // Scambia i valori.
                    //
                    scambio = lista[k];
                    lista[k] = lista[a];
                    lista[a] = scambio;
                  }
              }
            bsort (lista, a+1, z);
          }
        return lista;
    }
```

Torre di Hanoi

Il problema della torre di Hanoi è descritto nella sezione 62.5.3.

```
//
// java HanoiApp <n-anelli> <piolo-iniziale> <piolo-finale>
//
import java.lang.*; // predefinita
//
class HanoiApp
{
    //
    static void hanoi (int n, int p1, int p2)
    {
        if (n > 0)
          {
            hanoi (n-1, p1, 6-p1-p2);
            System.out.println ("Muovi l'anello " + n
                                + " dal piolo " + p1
                                + " al piolo " + p2 + ".");
            hanoi (n-1, 6-p1-p2, p2);
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int n;
        int p1;
        int p2;
        //
        n = Integer.valueOf(args[0]).intValue ();
        p1 = Integer.valueOf(args[1]).intValue ();
        p2 = Integer.valueOf(args[2]).intValue ();
        //
        hanoi (n, p1, p2);
    }
}
//
```

Quicksort

L'algoritmo del Quicksort è stato descritto nella sezione 62.5.4.

```
//
// java QSortApp
//
import java.lang.*; // predefinita
//
class QSortApp
{
    //
    static int part (int[] lista, int a, int z)
    {
        int scambio;
        //
        // Si assume che a sia inferiore a z.
        //
        int i = a + 1;
        int cf = z;
        //
        // Inizia il ciclo di scansione dell'array.
        //
        while (true)
          {
            while (true)
              {
                //
                // Sposta i a destra.
                //
                if ((lista[i] > lista[a]) || (i >= cf))
                  {
                    break;
```

```
                }
              else
                {
                  i++;
                }
            }
          while (true)
            {
              //
              // Sposta cf a sinistra.
              //
              if (lista[cf] <= lista[a])
                {
                  break;
                }
              else
                {
                  cf--;
                }
            }
          //
          if (cf <= i)
            {
              //
              // È avvenuto l'incontro tra i e cf.
              //
              break;
            }
          else
            {
              //
              // Vengono scambiati i valori.
              //
              scambio = lista[cf];
              lista[cf] = lista[i];
              lista[i] = scambio;
              //
              i++;
              cf--;
            }
        }
      //
      // A questo punto lista[a..z] è stata ripartita e cf è la
      // collocazione di lista[a].
      //
      scambio = lista[cf];
      lista[cf] = lista[a];
      lista[a] = scambio;
      //
      // A questo punto, lista[cf] è un elemento (un valore) nella
      // giusta posizione.
      //
      return cf;
    }
    //
    static int[] quicksort (int[] lista, int a, int z)
    {
        int cf;
        //
        if (z > a)
          {
            cf = part (lista, a, z);
            quicksort (lista, a, cf-1);
            quicksort (lista, cf+1, z);
          }
        //
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento. Qui si restituisce ugualmente un
        // riferimento all'array ordinato.
        //
        return lista;
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Ordina l'array.
        // In Java, gli array sono oggetti e come tali vengono passati
        // per riferimento.
        //
        quicksort (lista, 0, args.length-1);
        //
        // Visualizza il risultato.
        //
        for (i = 0; i < lista.length; i++)
          {
            System.out.println ("lista[" + i + "] = "
                                + lista[i]);
          }
    }
}
```



```
//
```

Permutazioni

L'algoritmo ricorsivo delle permutazioni è descritto nella sezione 62.5.5.

```
//
// java PermutaApp
//
import java.lang.*; // predefinita
//
class PermutaApp
{
    //
    static void permuta (int[] lista, int a, int z)
    {
        int scambio;
        int k;
        int i;
        //
        // Se il segmento di array contiene almeno due elementi, si
        // procede.
        //
        if ((z - a) >= 1)
          {
            //
            // Inizia un ciclo di scambi tra l'ultimo elemento e uno
            // degli altri contenuti nel segmento di array.
            //
            for (k = z; k >= a; k--)
              {
                //
                // Scambia i valori.
                //
                scambio = lista[k];
                lista[k] = lista[z];
                lista[z] = scambio;
                //
                // Esegue una chiamata ricorsiva per permutare un
                // segmento più piccolo dell'array.
                //
                permuta (lista, a, z-1);
                //
                // Scambia i valori.
                //
                scambio = lista[k];
                lista[k] = lista[z];
                lista[z] = scambio;
              }
          }
        else
          {
            //
            // Visualizza la situazione attuale dell'array.
            //
            for (i = 0; i < lista.length; i++)
              {
                System.out.print (" " + lista[i]);
              }
            System.out.println ("");
          }
    }
    //
    // Inizio del programma.
    //
    public static void main (String[] args)
    {
        int[] lista = new int[args.length];
        int i;
        //
        // Conversione degli argomenti della riga di comando in
        // numeri.
        //
        for (i = 0; i < args.length; i++)
          {
            lista[i] = Integer.valueOf(args[i]).intValue ();
          }
        //
        // Esegue le permutazioni.
        //
        permuta (lista, 0, args.length-1);
    }
}
//
```

Parte xxix

Scheme

Scheme: preparazione 1039

MIT Scheme 1039
Kawa 1041
Riferimenti 1044

Scheme: introduzione 1045

Aspetto generale 1045
Tipi di dati 1047
Costanti letterali 1049
Espressioni 1050
Funzioni comuni nelle espressioni e particolarità di alcuni tipi di dati elementari 1052
Strutture di controllo 1058
Conclusione di un programma Scheme 1062
Riferimenti 1062

Scheme: struttura del programma e campo di azione 1063

Definizione e campo di azione 1063
Definizione «lambda» 1065
Ricorsione 1066
Funzioni «let», «let*» e «letrec» 1066

Scheme: liste e vettori 1069

Liste e coppie 1069
Vettori 1073
Strutture di controllo applicate alle liste 1073
Riferimenti 1074

Scheme: I/O 1075

Apertura e chiusura 1075
Ingresso dei dati 1075
Uscita dei dati 1076

Scheme: esempi di programmazione 1079

Problemi elementari di programmazione 1079
Scansione di array 1084
Algoritmi tradizionali 1086

MIT Scheme 1039
Utilizzo interattivo 1039
REPL: l'ambito di funzionamento 1040
Utilizzo non interattivo 1040
Compilazione e caricamento di file 1040
Kawa 1041
Utilizzo interattivo 1041
Avvio dell'interprete Kawa 1042
Compilazione 1042
Riferimenti 1044

Scheme è un linguaggio di programmazione discendente dal LISP, inventato da Guy Lewis Steele Jr. e Gerald Jay Sussman nel 1995 presso il MIT. Scheme è importante soprattutto in quanto lo si ritrova utilizzato in situazioni estranee alla realizzazione di programmi veri e propri; in particolare, i fogli di stile DSSSL utilizzano il linguaggio Scheme.

In questo capitolo vengono mostrati gli strumenti più comuni che possono essere utilizzati per fare pratica con questo linguaggio di programmazione: MIT Scheme e Kawa, entrambi interpreti Scheme.

MIT Scheme

L'interprete Scheme del MIT [1] è disponibile per varie piattaforme. La versione per GNU/Linux può essere ottenuta dal MIT, a partire all'indirizzo *http://www.swiss.ai.mit.edu/projects/scheme/*. Il pacchetto può essere estratto a partire da una directory temporanea, da dove poi viene avviato uno script che provvede all'installazione, solitamente a partire dalla gerarchia '`/usr/local/`':

```
# tar xzvf scheme-7.5/scheme-7.5.12-ix86-gnu-linux.tar.gz
```
[*Invio*]

```
# cd dist-7.5 [ Invio ]

# ./install.sh [ Invio ]
```

Nel sito in cui viene distribuito questo interprete, si trova anche la documentazione per il suo utilizzo. Qui si intende mostrare solo l'essenziale.

Utilizzo interattivo

Per avviare l'interprete Scheme del MIT, basta il comando seguente:

```
$ scheme [ Invio ]
```

Quello che si vede dopo è una presentazione, seguita dall'invito a inserire comandi secondo il linguaggio Scheme.

```
Scheme saved on Sunday October 18, 1998 at 11:02:46 PM
  Release 7.4.7
  Microcode 11.151
  Runtime 14.168

1 ]=>
```

Per verificare rapidamente il funzionamento, basta utilizzare un'istruzione Scheme elementare che permette la visualizzazione di un messaggio:

```
1 ]=> (display "Ciao mondo!") [ Invio ]

Ciao mondo!
;Unspecified return value
```

Quello che si ottiene, come si vede, è l'emissione del messaggio, seguito dalla conferma che l'istruzione non ha restituito alcun valore.

La conclusione di una sessione di lavoro con l'interprete, si ottiene con l'istruzione '**`(exit)`**', dopo la quale viene richiesta una conferma, a cui si risponde con la lettera '**`y`**':

```
1 ]=> (exit) [ Invio ]
```

```
Kill Scheme (y or n)? y

  Happy Happy Joy Joy.
```

REPL: l'ambito di funzionamento

REPL sta per *Read-eval-print loop* e rappresenta una struttura di sottosessioni di lavoro all'interno dell'interprete. Il testo che appare come invito a inserire delle istruzioni, indica un numero intero positivo che rappresenta il livello REPL:

```
1 ]=>
```

Inizialmente questo livello è il primo, cioè il numero uno, ma può aumentare quando per qualche motivo c'è bisogno di passare a una sottosessione. La situazione tipica per la quale si passa a un livello successivo è l'inserimento di un'istruzione errata. Si osservi l'esempio seguente, in cui per errore non è stata delimitata la stringa che si vuole visualizzare:

```
1 ]=> (display Ciao mondo!) [Invio]

  ;Unbound variable: mondo!
  ;To continue, call RESTART with an option number:
  ; (RESTART 3) => Specify a value to use instead of mondo!.
  ; (RESTART 2) => Define mondo! to a given value.
  ; (RESTART 1) => Return to read-eval-print level 1.
```

A seguito di questo, si osserva che la stringa di invito è cambiata, indicando il passaggio a un secondo livello, a causa di un errore. Generalmente, per tornare al primo livello basta l'istruzione '**`(restart 1)`**', come si vede chiaramente nella spiegazione.

```
2 error> (restart 1) [Invio]
```

Se si fanno altri errori, si passa a livelli successivi, dai quali è possibile tornare sempre al primo livello nel modo appena mostrato.

Utilizzo non interattivo

Un programma Scheme può essere interpretato direttamente avviando '**`scheme`**' nel modo seguente:

```
scheme < sorgente_scheme
```

In pratica, si fornisce il sorgente attraverso lo standard input, come se venisse digitato attraverso la tastiera.

Compilazione e caricamento di file

L'interprete Scheme del MIT, consente anche una sorta di compilazione, con la quale si genera un formato intermedio, più pratico per l'esecuzione. Per ottenere questo, occorre avviare l'eseguibile '**`scheme`**' con l'opzione '**`-compiler`**'.

```
$ scheme -compiler [Invio]
```

Una volta predisposto un sorgente Scheme, lo si può compilare attraverso l'interprete con l'istruzione seguente:

```
(cf file_sorgente [file_destinazione])
```

Come si vede, l'indicazione di un file di destinazione è facoltativa, dal momento che in mancanza di questa, si utilizza un nome con la stessa radice di quello del sorgente.

```
1 ]=> (cf "ciao_mondo.scm") [Invio]
```

L'esempio mostra la compilazione del sorgente '`ciao_mondo.scm`', per generare il file '`ciao_mondo.com`'. La stessa cosa avrebbe potuto essere ottenuta con una delle due istruzioni seguenti:

```
1 ]=> (cf "ciao_mondo.scm" "ciao_mondo") [Invio]

1 ]=> (cf "ciao_mondo.scm" "ciao_mondo.com") [Invio]
```

La compilazione di questo tipo può essere utile per memorizzare dei sottoprogrammi da caricare durante una sessione interattiva. L'esempio seguente è la dichiarazione della funzione '**`fattoriale`**', il cui scopo è quello di calcolare il fattoriale di un numero intero.


```
(define (fattoriale n)
    (if (= n 0)
        1
        (* n (fattoriale (- n 1)))))
```

Il sorgente contenente esclusivamente queste righe, potrebbe chiamarsi '`fattoriale.scm`' ed essere stato compilato generando il file '`fattoriale.com`'.

L'interprete consente di caricare un file sorgente, o uno compilato, attraverso l'istruzione seguente:

```
(load file)
```

Se il nome del file viene indicato per intero, viene caricato in modo preciso quel file, mentre se si omette l'estensione, l'interprete cerca prima di trovare un file con l'estensione '`.com`', preferendo così una versione compilata eventuale.

Il caricamento di un file coincide anche con la sua esecuzione; se si tratta di dichiarazioni di variabili o di funzioni, si può avere la sensazione che non sia accaduto nulla. In questo caso, caricando il file '`fattoriale.com`', oppure '`fattoriale.scm`', si ottiene la disponibilità della funzione '**`fattoriale`**':

```
1 ]=> (load "fattoriale.scm") [Invio]

  ;Loading "fattoriale.scm" -- done
  ;Value: fattoriale

1 ]=> (fattoriale 3) [Invio]

  ;Value: 6
```

Kawa

Kawa [2] è un sistema Scheme, scritto in Java, in grado di funzionare come interprete e anche come compilatore, per trasformare un sorgente Scheme in un binario Java.

Come si può intendere, per poter utilizzare Kawa occorre avere installato Java (il JDK o Kaffe, come descritto nel capitolo u122). Di solito, per utilizzare Kawa come interprete, è sufficiente il comando '**`kawa`**', che dovrebbe corrispondere a uno script in grado di avviare l'interpretazione Java di '`repl.class`', che a sua volta dovrebbe trovarsi nella directory '`/usr/share/java/kawa/repl.class`'. Eventualmente, dovendo fare a meno di questo script, basterebbe il comando seguente:

```
$ java kawa.repl [Invio]
```

A ogni modo, questo non basta, dal momento che Kawa dispone di una propria libreria di classi che va aggiunta tra i percorsi della variabile di ambiente '**`CLASSPATH`**'. Lo script a cui si faceva riferimento, dovrebbe essere già predisposto in modo tale da includere in questa variabile di ambiente anche il percorso per la libreria di classi di Kawa, tuttavia, volendo realizzare dei binari Java indipendenti, partendo da programmi Scheme, è necessario pubblicizzare tale libreria anche all'esterno dell'interprete Kawa.

Le istruzioni seguenti sono adatte a una shell Bourne, o a una sua derivata e fanno riferimento all'ipotesi che la libreria di classi di Kawa sia stata installata a partire dalla directory '`/usr/share/java/`' (in pratica, si intende che in questo caso la libreria sia stata estratta dal solito archivio compresso):

```
CLASSPATH="$CLASSPATH:/usr/share/java/"
export CLASSPATH
```

Utilizzo interattivo

Per avviare l'interprete Scheme di Kawa, basta il comando seguente:

```
$ kawa [Invio]
```

Oppure, in mancanza di questo script:

```
$ java kawa.repl [Invio]
```

In questo secondo caso, è indispensabile la predisposizione della variabile di ambiente '**`CLASSPATH`**'. Quello che si vede dopo è un invito a inserire delle istruzioni Scheme:

```
#|kawa:1|#
```

Anche con l'interprete Kawa, si può fare una verifica rapida del funzionamento, utilizzando l'istruzione '**`display`**':

```
#|kawa:1|# (display "Ciao mondo!") (newline) [Invio]

Ciao mondo!
#|kawa:2|#
```

Mano a mano che si inseriscono delle istruzioni, il numero che compone il testo dell'invito si incrementa progressivamente, indipendentemente dal fatto che siano stati fatti degli errori.

Anche con Kawa, la conclusione di una sessione di lavoro con l'interprete si ottiene con l'istruzione '**`(exit)`**':

```
#|kawa:2|# (exit) [Invio]
```

Avvio dell'interprete Kawa

Lo script '**`kawa`**', ovvero il comando '**`java kawa.repl`**', permette l'utilizzo di alcune opzioni che possono rivelarsi importanti.

```
kawa [opzioni]
```

In particolare, l'interprete Kawa può leggere ed eseguire le istruzioni contenute in un file apposito, '`~/.kawarc.scm`', prima di procedere con le attività normali. Il file in questione è semplicemente un sorgente Scheme.

Tabella u126.11. Alcune opzioni.

Opzione	Descrizione
`-e` ***espressione***	Fa sì che Kawa valuti l'espressione, interpretandola come una serie di istruzioni Scheme, senza leggere il file '`~/.kawarc.scm`'.
`-c` ***espressione***	Fa sì che Kawa valuti l'espressione, interpretandola come una serie di istruzioni Scheme, dopo aver letto ed eseguito il contenuto del file '`~/.kawarc.scm`'.
`-f` ***file_sorgente_scheme***	Fa in modo che Kawa legga ed esegua il contenuto del file indicato come argomento, ignorando il file di configurazione. Se al posto del nome si indica un trattino orizzontale ('`-`'), si intende specificare lo standard input.
`-C` ***file_sorgente_scheme***	Compila il sorgente indicato in una classe Java. Se si vuole generare un programma autonomo, occorre utilizzare anche l'opzione '**`--main`**'.
`--main`	Assieme all'opzione '**`-C`**', consente di generare un binario Java, autonomo.

Segue la descrizione di alcuni esempi.

- `$` **`kawa -c '(display "Ciao mondo!") (newline)'`** [*Invio*]

  Visualizza il messaggio «Ciao mondo!», senza prendere in considerazione il file di configurazione.

- `$` **`kawa -f ciao_mondo.scm`** [*Invio*]

  Esegue il contenuto del file '`ciao_mondo.scm`', che si presume essere un sorgente Scheme.

Compilazione

Con Kawa è possibile compilare sia all'interno della sessione di lavoro dell'interprete, sia all'esterno. Nel primo caso, si utilizza l'istruzione seguente:

```
(compile-file nome_sorgente radice_destinazione)
```

Con questa si può fare qualcosa del genere:

```
#|kawa:m|# (compile-file "ciao_mondo.scm" "ciao") [Invio]
```


In tal modo, dal file sorgente '`ciao_mondo.scm`' si ottiene il file '`ciao.zip`', contenente una classe non meglio precisata, il quale può essere ricaricato successivamente con l'istruzione

```
(load radice_file_compilato)
```

In pratica, volendo caricare ed eseguire il contenuto del file '`ciao.zip`', basta l'istruzione seguente:

```
#|kawa:m|# (load "ciao") [Invio]
```

La compilazione al di fuori dell'ambiente interattivo, si ottiene utilizzando l'opzione '**`-C`**', con la quale si ottengono delle classi Java non compresse. Si distinguono due situazioni:

```
kawa [altre_opzioni] -C sorgente_scheme
```

```
kawa [altre_opzioni] --main -C sorgente_scheme
```

Nel primo caso si ottiene un file con estensione '`.class`' che può essere caricato all'interno di una sessione di lavoro dell'interprete, con la funzione '**`load`**' già mostrata; nel secondo si ottiene un programma indipendente.

A titolo di esempio, si può utilizzare il sorgente di prova che viene mostrato di seguito:

```
;
; fattoriale.scm
;
(define (fattoriale n)
    (if (= n 0)
        1
        (* n (fattoriale (- n 1)))))
```

Questo può essere compilato in modo da poterlo ricaricare successivamente:

```
$ kawa -C fattoriale.scm [Invio]
```

Si ottiene il file '`fattoriale.class`'. All'interno dell'interprete, si può caricare quanto compilato con la funzione '**`load`**' (con la quale si potrebbe caricare anche un sorgente non compilato, indicando il nome completo del file).

```
#|kawa:m|# (load "fattoriale") [Invio]
```

Quindi si potrebbe sfruttare la funzione '**`fattoriale`**' dichiarata all'interno del file appena caricato:

```
#|kawa:n|# (display (fattoriale 3)) (newline) [Invio]

6
```

Volendo rendere autonomo il programma del calcolo del fattoriale, occorrerebbe aggiungere le istruzioni necessarie per gestire l'inserimento e l'emissione dei dati:

```
;
; fattoriale.scm
;
(define (fattoriale n)
    (if (= n 0)
        1
        (* n (fattoriale (- n 1)))))

(define valore 0)
(display "Inserisci un numero intero: ")
(set! valore (read))
(display "Il fattoriale di ")
(display valore)
(display " è ")
(display (fattoriale valore))
(newline)
```

Per la sua compilazione si procede nel modo già descritto, utilizzando l'opzione '**`--main`**':

```
$ kawa --main -C fattoriale.scm [Invio]
```

Anche in questo caso si genera il file '`fattoriale.class`', che però può essere avviato direttamente da Java:

```
$ java fattoriale [Invio]

Inserisci un numero intero: 3 [Invio]
```

```
Il fattoriale di 3 è 6
```

Riferimenti

- *MIT Scheme*
  http://www.swiss.ai.mit.edu/projects/scheme/
  ftp://swiss-ftp.ai.mit.edu/pub/
- Per Bothner, *Kawa, the Java-based Scheme system*, 1999
  http://www.gnu.org/software/kawa/
  ftp://ftp.gnu.org/pub/gnu/kawa/

[1] **MIT Scheme** GNU GPL

[2] **Kawa** GNU GPL, ma con meno restrizioni

Scheme: introduzione

Aspetto generale 1045
Identificatori e convenzioni nei nomi 1046
Funzioni o procedure 1047
Tipi di dati 1047
Costanti letterali 1049
Costanti booleane 1049
Costanti numeriche 1049
Stringhe 1049
Costanti carattere 1050
Espressioni 1050
Riferimenti variabili 1050
Espressioni letterali 1050
Ordine nella valutazione di un'espressione 1051
Funzioni comuni nelle espressioni e particolarità di alcuni tipi di dati elementari 1052
Numeri 1052
Valori logici, funzioni di confronto e funzioni logiche 1055
Caratteri 1057
Stringhe 1057
Strutture di controllo 1058
Funzione «begin» 1058
Struttura condizionale: «if» 1059
Struttura di selezione: «cond» 1059
Struttura di selezione: «case» 1060
Iterazione: «do» 1061
Conclusione di un programma Scheme 1062
Riferimenti 1062

Il linguaggio Scheme ha una filosofia che si basa fondamentalmente sul suo tipo di notazione. Scheme è un linguaggio utile per rappresentare un problema, più che per realizzare un programma completo. La standardizzazione di questo linguaggio è riferita fondamentalmente a un documento che viene aggiornato periodicamente: R^{n}RS, ovvero *Revised-n Report on the Algorithmic Language Scheme*, dove ***n*** è il numero di questa revisione (attualmente dovrebbe trattarsi della quinta). Tuttavia, la standardizzazione riguarda gli aspetti fondamentali del linguaggio, mentre ogni realizzazione che utilizza Scheme introduce le estensioni necessarie alle circostanze.

In questo capitolo si vogliono descrivere solo alcuni degli aspetti più importanti di questo linguaggio. Il documento di riferimento è quello citato, ovvero *R^{5}RS*; alla fine del capitolo si possono trovare anche altri riferimenti per guide più o meno dettagliate su Scheme.

Aspetto generale

Il linguaggio Scheme prevede dei commenti, che vengono ignorati regolarmente: si distinguono perché iniziano con un punto e virgola (‘`;`’) e terminano alla fine della riga. Generalmente, le righe vuote e quelle bianche sono ignorate nello stesso modo. In generale, le istruzioni Scheme hanno l'aspetto di qualcosa che è racchiuso tra parentesi tonde.

```
(display "Ciao")
```

Per comprenderne il senso, l'esempio precedente potrebbe essere espresso come si vede sotto, se lo si volesse rappresentare in un linguaggio ipotetico, basato sulle funzioni:

```
display ("Ciao")
```

Tutto quello che si fa con Scheme viene ottenuto attraverso chiamate di funzione, ovvero, secondo la terminologia utilizzata da *R*^{5}*RS*, ***procedure***, che possono restituire o meno un valore. Le chiamate di queste procedure, o di queste funzioni, iniziano con un nome, posto subito dopo la parentesi tonda di apertura, continuando eventualmente con l'elenco dei parametri che gli vengono passati, separati semplicemente da uno o più spazi, anche verticali (non si utilizzano virgole o altri simboli di interpunzione), terminando con la parentesi tonda di chiusura.

(*nome* [*parametro_1* [*parametro_2*] ... [*parametro_n*]])

Da quanto affermato, si intende anche che un'istruzione può essere interrotta in qualunque punto in cui potrebbe essere inserito uno spazio, per riprenderla nella riga successiva, incolonnandola in base allo stile preferito. Si osservi l'esempio seguente:

```
(+ 3 4)
```

si tratta di una chiamata a una funzione denominata '**+**', a cui vengono passati i parametri '**3**' e '**4**'. Si intende, intuitivamente, che questa funzione restituisca la somma dei parametri.

Le istruzioni non hanno bisogno di essere terminate da un qualche simbolo di interpunzione, dal momento che le parentesi tonde esprimono chiaramente l'estensione di queste e l'ambito relativo all'interno dei vari annidamenti.

Questo tipo di notazione ha diversi pregi, ma ha il difetto fondamentale di essere un po' difficile da seguire visivamente, soprattutto a causa dell'affollarsi delle parentesi tonde.

In questi capitoli si cerca di utilizzare un allineamento di queste parentesi che renda più facile la lettura delle istruzioni, anche se si tratta di uno stile che di solito non si applica.

Per facilitare la comprensione degli esempi, in questi capitoli dedicati a Scheme, si utilizza il simbolo '**===>**' per indicare il valore restituito da una funzione (che appare alla sua destra).

Identificatori e convenzioni nei nomi

I nomi utilizzati per «identificare» qualunque cosa in Scheme, possono essere scritti utilizzando le lettere dell'alfabeto, le cifre numeriche e una serie di caratteri particolari che vengono considerati come un'estensione ai caratteri alfabetici:

```
!  $  %  &  *  +  -  .  /  :  <  =  >  ?  @  ^  _
```

Non tutte le combinazioni sono possibili: in generale non è ammissibile che tali nomi inizino con una cifra numerica.

In generale, Scheme non dovrebbe fare differenza tra lettere maiuscole e minuscole nei nomi che identificano qualcosa.

È importante osservare che, a differenza di altri linguaggi di programmazione, caratteri come '**+**', '**-**', '*****' e '**/**', possono essere (e in pratica sono) dei nomi.

```
(+ 3 4)
```

Come è già stato fatto osservare, l'esempio mostra la chiamata della funzione (procedura) '**+**', a cui vengono passati i valori tre e quattro come parametri.

Anche se si possono usare caratteri insoliti nei nomi degli identificatori, quando si dichiara qualcosa, come il nome di una variabile, o di una funzione, è bene astenersi dalle cose troppo stravaganti, a meno che ci sia un buon motivo per le scelte che si fanno. In generale, sono già stabilite delle convenzioni per i nomi delle funzioni, almeno quelle che fanno già parte del linguaggio standard:

- le funzioni il cui nome termina con un punto interrogativo ('**?**') sono intese essere dei «predicati», ovvero delle funzioni che verificano l'avverarsi di una condizione (la verità di un'affermazione) e restituiscono un valore booleano;
- le funzioni il cui scopo è quello di modificare il valore di una variabile, senza cambiarne l'allocazione (per la precisione si tratta di modificare un valore in un'area di memoria già allocata), terminano con un punto esclamativo ('**!**');
- Le funzioni il cui scopo è quello di convertire un «oggetto» di un tipo, in un altro di tipo differente, contengono un '**->**' all'interno del nome.

Per permettere di comprendere meglio come possa essere formato un identificatore, si osservi l'elenco seguente di nomi, che rappresentano tutti delle possibilità valide:

```
ciao            a               b               +               -
*               list->vector    ABCdef123       A123b124        <=?
ciao-come-stai-io-sto-bene-grazie
```

Funzioni o procedure

Scheme è un linguaggio basato sulle funzioni, per quanto queste vengano chiamate «procedure» nella sua terminologia specifica. Questo significa, per esempio, che tutte le espressioni che si possono scrivere con Scheme sono dei valori costanti, oppure delle chiamate di funzione, più o meno annidate. Anche le strutture di controllo sono realizzate in forma di funzione.

È importante osservare che in Scheme non esiste una funzione principale che debba essere eseguita prima delle altre; si segue semplicemente l'ordine sequenziale in cui appaiono le istruzioni. In generale, con lo stesso criterio, le funzioni che si utilizzano devono essere state dichiarate prima del loro utilizzo.

Tipi di dati

Scheme utilizza una gestione speciale per le variabili. La dichiarazione di una variabile implica l'allocazione di uno spazio di memoria adatto e l'abbinamento del puntatore relativo a una variabile.

(define *variabile* [*valore_iniziale*])

L'esempio seguente, alloca l'area di memoria necessaria a contenere un numero intero, quindi abbina all'identificatore '**x**' il puntatore a questa area.

```
(define x 1)
```

In pratica, l'identificatore '**x**' si comporta come una variabile di un linguaggio di programmazione «normale», dal momento che quando viene valutato in un'espressione restituisce esattamente il valore a cui punta.

Questa distinzione, non è soltanto una questione di pignoleria, ma si tratta di un punto fondamentale della filosofia di Scheme: la dichiarazione successiva dello stesso identificatore, non va a modificare l'informazione precedente, ma alloca una nuova area di memoria. L'allocazione precedente non viene recuperata e potrebbe continuare a essere utilizzata da ciò che è stato dichiarato prima del cambiamento. In questo senso, a livello teorico, il linguaggio Scheme non prevede un sistema di eliminazione degli oggetti inutilizzati (lo spazzino, ovvero il *garbage collector*), benché le realizzazioni possano attuare in pratica queste forme di ottimizzazione quando sono in grado di provare che un'area di memoria allocata non può più essere presa in considerazione nel programma.

Proprio a causa di questa particolarità di Scheme, per assegnare un valore a un'area di memoria già allocata, attraverso l'identificatore relativo, si utilizza la funzione '**set!**':

(set! *variabile* *espressione_del_valore_da_assegnare*)

Il punto esclamativo finale che compone il nome della funzione, serve a sottolineare il fatto che si ottiene la modifica di un valore già allocato, senza allocare un'altra area di memoria.

I dati secondo Scheme sono organizzati in ***oggetti***, ma non nel senso che viene attribuito dai linguaggi di programmazione a oggetti (*object oriented*). I tipi di dati di Scheme sono precisamente:

- booleano -- inteso come il risultato di un'espressione logica, o una costante booleana;
- coppia (lista non vuota);
- simbolico -- che fa riferimento a costanti simili alle stringhe, ma che sono trattate diversamente in Scheme;
- numerico;
- carattere -- un carattere singolo che non è una stringa;
- stringa;
- vettore -- quello che per gli altri linguaggi è un array;
- porta, o flusso -- ovvero un file aperto;
- procedura -- le funzioni di Scheme.

I dati hanno una loro essenza e una loro rappresentazione esterna, che corrisponde al modo in cui vengono espressi a livello umano. Questa rappresentazione può consentire a volte l'uso di forme diverse ed equivalenti; per esempio, il numero 16 può essere espresso con la sequenza dei caratteri '**16**', oppure '**#d16**', '**#x10**' e in altri modi ancora.

Tuttavia, è bene osservare che un oggetto per Scheme può essere di un tipo solo. Si parla in questo senso di «tipi disgiunti».

Scheme fornisce alcuni predicati, ovvero alcune funzioni, per il controllo del tipo a cui appartiene un oggetto. Nello stesso ordine in cui sono stati elencati i tipi di dati, si tratta di: '**boolean?**', '**pair?**', '**symbol?**', '**number?**', '**char?**', '**string?**', '**vector?**', '**port?**', '**procedure?**'. Per esempio, l'istruzione seguente restituisce *Vero* se l'identificatore '**x**' fa riferimento a un numero:

```
(number? x)
```

Tra tutti i tipi di dati visti, ne esiste uno speciale: la lista vuota, che non appartiene alle coppie. Per identificare una lista di qualunque tipo, includendo anche quelle vuote, si usa il predicato '**list?**'.

Tabella u127.9. Elenco dei predicati utili per verificare l'appartenenza ai vari tipi di dati.

Predicato	Descrizione
`(boolean?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un valore logico booleano.
`(pair?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato una «coppia» (lista non vuota).
`(list?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato una lista (anche vuota).
`(symbol?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un simbolo.
`(number?` ***espressione*** `)`	*Vero* se l'espressione dà un risultato numerico di qualunque tipo.
`(char?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un carattere.
`(string?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato una stringa.
`(vector?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un vettore.
`(port?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato una «porta».
`(procedure?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato una funzione.

Costanti letterali

Scheme ha una gestione particolare delle espressioni, dove al loro interno è speciale la gestione dei valori costanti. Questo fatto viene chiarito nel seguito. Tuttavia, è necessario conoscere subito in che modo possono essere indicati i valori più comuni in un sorgente Scheme.

Costanti booleane

I valori booleani possono essere rappresentati attraverso la sigla '**#t**' per *Vero* e '**#f**' per *Falso*.

Costanti numeriche

I valori numerici possono essere usati nel modo consueto, quando si tratta di valori interi (positivi o negativi), quando si vogliono indicare numeri che hanno una quantità fissa di decimali e quando si usa la notazione scientifica comune ('***x*e*y***').

```
67
+67
-67
678.67
+678.67
-678.67
6.7867e2
67867e-3
```

In aggiunta a quello che si può vedere dagli esempi mostrati sopra, si possono indicare dei valori specificando la base di numerazione. Per ottenere questo, si utilizza un prefisso del tipo:

```
#x
```

In questo caso, ***x*** è una lettera che esprime la base di numerazione. Segue l'elenco di questi prefissi:

- '**#b**' -- numero binario;
- '**#o**' -- numero ottale;
- '**#d**' -- numero decimale;
- '**#x**' -- numero esadecimale.

Per esempio, '**#x10**' è equivalente a '**#d16**', ovvero a 16 senza prefissi.

Scheme consente di utilizzare anche altri tipi di notazioni, per indicare alcuni tipi particolari di numeri. Questa caratteristica di Scheme viene descritta più avanti.

Stringhe

Scheme ha una gestione speciale delle espressioni costanti, cosa che viene descritta in seguito. Ugualmente, è prevista la presenza delle stringhe, rappresentate attraverso una sequenza di caratteri delimitata da una coppia di apici doppi: '**"**…**"**'.

All'interno delle stringhe è previsto l'uso di sequenze di escape composte dalla barra obliqua inversa ('\') seguita da un carattere. Secondo lo standard *R*^{5}*RS* è prevista solo la sequenza '**\"**', per inserire un apice doppio, e '\\', per poter inserire una barra obliqua inversa. Le varie realizzazioni di Scheme, possono prevedere l'utilizzazione di altre sequenze di escape, per esempio come avviene nel linguaggio C.

Potrebbe venire spontaneo l'utilizzo della sequenza '**\n**' per inserire il codice di interruzione di riga all'interno di una stringa; tuttavia, anche se potrebbe funzionare, Scheme dispone della funzione '**newline**', che non prevede l'uso di parametri, il cui scopo è quello di fare ciò che serve per ottenere un avanzamento all'inizio della riga successiva.

```
(display "ciao a tutti, si, proprio a \"tutti\"")
(newline)
```

Costanti carattere

In Scheme, i caratteri sono qualcosa di diverso dalle stringhe, ma questo vale anche per altri linguaggi di programmazione. Tuttavia, la rappresentazione di una costante carattere è molto diversa rispetto alle stringhe:

#\carattere | *#\nome_carattere*

Questi caratteri, sempre secondo Scheme, sono oggetti singoli e non possono essere uniti assieme a formare una stringa, a meno di utilizzare delle funzioni apposite di conversione in stringa. Segue un elenco che mostra alcuni esempi di rappresentazione di questi oggetti carattere.

- '**#\a**' -- la lettera «a» minuscola;
- '**#\A**' -- la lettera «A» maiuscola;
- '**#\(**' -- la parentesi tonda aperta;
- '**#\ **' -- lo spazio (dopo la barra obliqua inversa c'è esattamente un carattere <*SP*>;
- '**#\space**' -- lo spazio, espresso per nome;
- '**#\newline**' -- il codice di interruzione di riga.

Espressioni

Un'espressione è qualcosa che, per mezzo di una valutazione, fa qualcosa, oppure restituisce un qualche valore, o fa tutte e due le cose. Le espressioni sono cose che riguardano praticamente tutti i linguaggi di programmazione, ma Scheme ha una gestione particolare quando si vuole evitare che qualcosa venga trasformato da una valutazione.

In pratica, in Scheme si distinguono le ***espressioni letterali***, che sono delle espressioni che per qualche ragione, non devono essere elaborate nel modo consueto, ma passate così come sono in modo letterale.

Riferimenti variabili

Nella filosofia di Scheme non si hanno delle variabili vere e proprie, ma degli identificatori che fanno riferimento a delle zone di memoria allocate. Tuttavia, si può usare ugualmente il termine «variabile», se si fa attenzione a ricordare la particolarità di Scheme.

La valutazione di una variabile in Scheme genera la restituzione del valore contenuto nell'area di memoria a cui questa punta. Se si usa un interprete Scheme, come quelli descritti nel capitolo introduttivo di questa parte, si può osservare quanto descritto in modo molto semplice:

```
(define x 195)
x                       ===> 195
```

In pratica, l'espressione banale che consiste nell'indicare semplicemente l'identificatore di una variabile, genera la restituzione del valore che in precedenza gli è stato assegnato.

Espressioni letterali

In un linguaggio di programmazione qualunque, le espressioni letterali corrispondono alle costanti letterali, come i numeri, le stringhe e oggetti simili. In Scheme si aggiungono anche altri oggetti.

costante

'dato

(quote *dato*)

A parte le costanti letterali normali, le altre espressioni letterali si distinguono per essere precedute da un apostrofo iniziale ('**'**'), oppure (ed è la stessa cosa), per essere indicate come argomento della funzione '**quote**'.

Inizialmente è difficile comprendere il senso di questa notazione. Tuttavia, è importante riconoscere subito che non si tratta di stringhe, in quanto lo scopo per il quale esistono queste espressioni letterali, è proprio quello di evitare che vengano valutate prima del necessario. Si osservino gli esempi seguenti; in particolare, si suppone che esista una variabile '**a**' che faccia riferimento a una zona di memoria contenente il valore uno.

```
(quote a)               ===> a «simbolo»
'a                      ===> a «simbolo»
a                       ===> 1
```

```
(quote (+ 1 2))         ===> (+ 1 2)
'(+ 1 2)                ===> (+ 1 2)
(+ 1 2)                 ===> 3
```

```
(quote (quote a))       ===> (quote a)
''a                     ===> (quote a)
'a                      ===> a «simbolo»
```

```
(quote "a")             ===> "a" «stringa»
'"a"                    ===> "a" «stringa»
"a"                     ===> "a" «stringa»
```

```
(quote 1)               ===> 1
'1                      ===> 1
1                       ===> 1
```

```
(quote #t)              ===> #t
'#t                     ===> #t
#t                      ===> #t
```

```
(quote #\a)             ===> #\a «carattere»
'#\a                    ===> #\a «carattere»
#\a                     ===> #\a «carattere»
```

Nei primi esempi si fa riferimento a qualcosa che si identifica attraverso la lettera «a». '**(quote a)**', ovvero '**'a**', non è un carattere e non è una stringa: è un simbolo non meglio identificato; dipende dal programmatore il significato che questo può avere. Per semplificare le cose, si è immaginato che si trattasse di una variabile.

Tra gli esempi si vede la possibilità di indicare una funzione per la somma, '**(+ 1 2)**', come espressione costante. Ci sono situazioni in cui questo è necessario, per esempio quando una funzione deve essere passata come argomento di un'altra, mentre lo scopo non è quello di passare il risultato della valutazione della prima.

Le costanti letterali, come le stringhe, i numeri, i caratteri e i valori booleani, possono essere indicati come espressioni letterali; in tal modo il risultato non cambia, dal momento che la valutazione di tali costanti restituisce le costanti stesse.

Ci sono altri tipi di dati che possono essere indicati in forma di espressioni letterali, ma non sono stati mostrati gli esempi relativi perché questi tipi non sono ancora stati descritti. Tuttavia, il senso non cambia: si usano le espressioni letterali quando non si può lasciare che queste siano valutate.

Ordine nella valutazione di un'espressione

L'ordine in cui viene valutata un'espressione è relativamente semplice in Scheme, dal momento che non si utilizzano operatori simbolici e tutto è espresso in forma di funzioni. In generale, si valuta prima ciò che sta nella posizione più «interna», venendo mano a mano verso l'esterno.

```
(* 3 (+ 2 4))
```

L'esempio appena mostrato si risolve secondo la sequenza di operazioni elencate di seguito:

- '**3**' ===> '**3**'
- valutazione di '**(+ 2 4)**'
  - '**2**' ===> '**2**'
  - '**4**' ===> '**4**'
  - '**2+4**' ===> '**6**'
- '**3*6**' ===> '**18**'

Funzioni comuni nelle espressioni e particolarità di alcuni tipi di dati elementari

Nei linguaggi di programmazione comuni, le espressioni si avvalgono prevalentemente di operatori di vario tipo, tanto che gli operatori sono di per sé delle funzioni, più o meno celate. Con Scheme, questa ambiguità viene eliminata, dal momento che tutte le operazioni di un'espressione si svolgono per mezzo di funzioni. Le funzioni che vengono descritte in queste sezioni, sono quelle che vengono utilizzate più frequentemente nelle espressioni di Scheme.

Il valore restituito da una funzione può essere di tipo diverso a seconda degli operandi utilizzati. Di solito si fa l'esempio della somma di due interi che genera un risultato intero. Scheme ha una gestione particolare dei numeri, almeno a livello teorico, per cui questi vengono classificati in modo molto più sofisticato di quanto facciano i linguaggi di programmazione normali.

Nella sezione dedicata ai numeri, è assente la spiegazione riguardo al tipo numerico «complesso». Eventualmente si può consultare il documento *R*^{5}*RS* in cui questo argomento è affrontato.

Numeri

Con Scheme, i numeri sono gestiti a due livelli differenti: l'astrazione matematica e la realizzazione pratica. Dal punto di vista dell'astrazione matematica, si distinguono i livelli seguenti:

- numero generico;
- numero complesso;
- numero reale;
- numero razionale;
- numero intero.

In generale, un numero che appartiene a una classe inferiore, è anche un numero che può essere considerato appartenente a tutti i livelli superiori. Per esempio, un numero razionale è anche un numero reale ed è anche un numero complesso, ecc.

Scheme fornisce una serie di predicati (funzioni), per la verifica dell'appartenenza di un valore a un tipo di numero. L'elenco si vede nella tabella u127.21. In generale, queste funzioni restituiscono il valore *Vero* ('`#t`') nel caso in cui sia valida l'appartenenza presunta.

Tabella u127.21. Elenco dei predicati utili per verificare l'appartenenza ai vari tipi numerici.

Predicato	Descrizione
`(number?` ***espressione*** `)`	*Vero* se l'espressione dà un risultato numerico di qualunque tipo.
`(complex?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un numero complesso.
`(real?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un numero reale.
`(rational?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un numero razionale.
`(integer?` ***espressione*** `)`	*Vero* se l'espressione dà come risultato un numero intero.

Nel modo in cui si rappresenta un numero si indica implicitamente il tipo di questo. Tuttavia, se Scheme è in grado di conoscere una semplificazione nel modo di rappresentarne il valore, lo classifica automaticamente nella fascia inferiore relativa. Per esempio, se 4/2 viene mostrato come numero razionale, dal momento che è equivalente a due, è anche un intero puro e semplice. Gli esempi seguenti mostrano in che modo possono reagire i predicati per la verifica del tipo numerico. Si osservi in particolare la disponibilità della notazione ***m***/***n***, che permette di indicare agevolmente i numeri razionali:

```
(integer? 3)            ===> #t
(rational? 3)           ===> #t
(real? 3)               ===> #t
(complex? 3)            ===> #t
(number? 3)             ===> #t
```

```
(integer? 6/2)          ===> #t
(integer? 3/2)          ===> #f
(rational? 6/2)         ===> #t
(rational? 3/2)         ===> #t
```

```
(integer? 1.1)          ===> #f
(rational? 1.1)         ===> #t (dipende dalla realizzazione di Scheme)
(real? 1.1)             ===> #t
```

Secondo Scheme, i numeri sono ***esatti*** o ***inesatti***, a seconda di varie circostanze, che possono dipendere anche dalla realizzazione che si utilizza. In generale, un numero è esatto se è stato fornito attraverso una costante che di per sé è esatta (come un numero intero o un numero razionale), oppure se deriva da numeri esatti utilizzati in operazioni esatte. Si comprende intuitivamente che nel momento in cui si introducono approssimazioni di qualche tipo, per qualche ragione, i valori che si ottengono dai calcoli che si fanno, non sono precisi, ma sono, appunto, inesatti. Nonostante sia molto facile generare risultati inesatti, anche quando si parte da valori esatti, ci sono alcune situazioni in cui i risultati sono esatti anche se i valori di partenza sono inesatti; per esempio, la moltiplicazione per uno zero esatto, genera uno zero esatto, qualunque sia l'altro valore. A proposito dell'esattezza o meno dei valori, sono disponibili alcune funzioni che sono elencate nella tabella u127.25.

Tabella u127.25. Elenco dei predicati e delle altre funzioni riferite ai valori esatti e inesatti.

Funzione	Descrizione
`(exact?` ***espressione*** `)`	*Vero* se l'espressione dà un risultato numerico esatto.
`(inexact?` ***espressione*** `)`	*Vero* se l'espressione dà un risultato numerico inesatto.
`(exact->inexact` ***espressione*** `)`	Converte il risultato dell'espressione in un valore numerico inesatto.
`(inexact->exact` ***espressione*** `)`	Converte il risultato dell'espressione in un valore numerico esatto.

Seguono alcuni esempi sull'uso di queste funzioni:

```
(exact? 3)              ===> #t
(exact? 3/2)            ===> #t
(exact? 1.5)            ===> #f
(exact->inexact 3)      ===> 3.0
(inexact->exact 1.5)    ===> 3/2
```

Come accennato all'inizio, oltre all'astrazione matematica si pone il problema della precisione dei valori inesatti (quelli che per altri linguaggi di programmazione sono semplicemente dei valori a virgola mobile). Ammesso che la realizzazione di Scheme permetta di distinguere tra diversi livelli di precisione, si possono rappresentare delle costanti numeriche «reali» (a virgola mobile), utilizzando la notazione esponenziale, dove al posto della lettera «e» consueta, si utilizzano rispettivamente le lettere, '`s`', '`f`', '`d`' e '`l`', che indicano valori a precisione ridotta (*short*), a singola precisione (*float*), a doppia precisione (*double*) e a precisione ancora maggiore (*long*).

Tabella u127.27. Elenco delle funzioni matematiche comuni.

Funzione	Descrizione
`(+` ***op***`...)`	Somma gli argomenti.
`(*` ***op***`...)`	Moltiplica gli argomenti.
`(-` ***op***`)`	Moltiplica il valore dell'operando per `-1`.
`(-` ***op1 op2***`...)`	Sottrae dal primo la somma degli operandi successivi.
`(/` ***op***`)`	Divide il primo operando per 1.
`(/` ***op1 op2***`...)`	Divide il primo operando per il secondo, divide il risultato per il terzo...
`(log` ***op***`)`	Calcola il logaritmo naturale.
`(exp` ***op***`)`	Calcola l'esponente.

Funzione	Descrizione
`(sin op)`	Calcola il seno.
`(cos op)`	Calcola il coseno.
`(tan op)`	Calcola la tangente.
`(asin op)`	Calcola l'arco-seno.
`(acos op)`	Calcola l'arco-coseno.
`(atan op)`	Calcola l'arco-tangente.
`(sqrt op)`	Calcola la radice quadrata.
`(expt op1 op2)`	Eleva il primo operando alla potenza del secondo.
`(abs op)`	Calcola il valore assoluto.
`(quotient op1 op2)`	Divide il primo operando per il secondo e restituisce il valore intero.
`(remainder op1 op2)`	Resto della divisione del primo operando per il secondo.
`(modulo op1 op2)`	Calcola il modulo (vedere nota).
`(ceiling op)`	Calcola la parte intera per eccesso.
`(floor op)`	Calcola la parte intera per difetto.
`(round op)`	Calcola la parte intera più vicina.
`(truncate op)`	Calcola la parte intera eliminando semplicemente la parte decimale.
`(max op…)`	Restituisce il valore massimo dei suoi operandi.
`(min op…)`	Restituisce il valore minimo dei suoi operandi.
`(gcd n_intero…)`	Calcola il massimo comune divisore dei vari operandi.
`(lcd n_intero…)`	Calcola il minimo comune multiplo dei vari operandi.
`(numerator n_razionale)`	Restituisce il numeratore di un numero razionale.
`(denominator n_razionale)`	Restituisce il denominatore di un numero razionale.

La tabella u127.27 riporta l'elenco delle funzioni più comuni che possono essere usate nelle espressioni aritmetiche e matematiche. In particolare si deve osservare che '**`remainder`**' e '**`modulo`**' si comportano nello stesso modo, tranne quando si utilizzano valori negativi (per approfondire la differenza si può leggere il documento di riferimento su Scheme, ovvero *R*^{5}*RS*).

Scheme permette di utilizzare più di due operandi per le funzioni che sommano, sottraggono, dividono e moltiplicano. A parte la spiegazione sintetica data nella tabella in cui sono state presentate, si può intendere il senso del loro funzionamento immaginando che le operazioni avvengono in modo progressivo, da sinistra a destra:

```
(- 5 3 2)
```

L'esempio appena mostrato equivale a:

```
(- (- 5 3) 2)
```

Nello stesso modo, si osservi l'esempio seguente:

```
(/ 5 3 2)
```

Questo equivale a:

```
(/ (/ 5 3) 2)
```

Infine, la tabella u127.32 riporta alcuni predicati utili per classificare in qualche modo un valore numerico.

Tabella u127.32. Elenco di altri predicati utili per classificare i valori numerici.

Funzione	Descrizione
`(zero? op)`	*Vero* se l'operando equivale a zero.
`(positive? op)`	*Vero* se l'operando è un numero positivo.
`(negative? op)`	*Vero* se l'operando è un numero negativo.
`(odd? op)`	*Vero* se l'operando è un numero dispari.
`(even? op)`	*Vero* se l'operando è un numero pari.

Scheme dispone di altre risorse per la gestione dei valori numerici; inoltre, ciò che è stato presentato qui è descritto in modo approssimativo. Se si vogliono sfruttare bene tali possibilità di questo linguaggio, è indispensabile studiare bene il documento *R*^{5}*RS*, già citato più volte, del quale si trova un riferimento alla fine del capitolo.

Valori logici, funzioni di confronto e funzioni logiche

Sono già state presentate le costanti booleane '**`#t`**' e '**`#f`**', che valgono per *Vero* e *Falso* rispettivamente. Per Scheme, da un punto di vista logico-booleano, valgono come *Vero* anche le liste (che vengono descritte in seguito), compresa la lista vuota, i simboli, i numeri, le stringhe, i vettori e le funzioni. In pratica, qualsiasi oggetto diverso dal tipo booleano, assieme al valore booleano '**`#t`**', vale come *Vero*, mentre solo '**`#f`**' vale per *Falso*. Tuttavia, per verificare che un oggetto corrisponda effettivamente a un valore booleano, si può usare il predicato seguente:

```
(boolean? oggetto)
```

Questo restituisce *Vero* in caso affermativo.

Alcune realizzazioni più vecchie di Scheme trattano la lista vuota, che si rappresenta con '**`()`**', come equivalente al valore booleano *Falso*.

Gli operatori logici sono realizzati in Scheme attraverso funzioni. La tabella u127.33 elenca queste funzioni.

Tabella u127.33. Elenco delle funzioni logiche.

Funzione	Descrizione
`(not op)`	Inverte il risultato logico dell'operando.
`(and op1 op2…)`	*Vero* se tutti gli operandi restituiscono *Vero*.
`(or op1 op2…)`	*Vero* se anche solo un operando restituisce *Vero*.

Per quanto riguarda il confronto, si distinguono situazioni diverse, a seconda che si vogliano confrontare dei valori numerici, carattere, stringa, oppure che si vogliano confrontare gli «oggetti». Le tabelle u127.34, u127.36, u127.38 e u127.40, riepilogano le funzioni in grado di eseguire tali confronti.

Tabella u127.34. Elenco delle funzioni per il confronto numerico.

Funzione	Descrizione
`(= op1 op2…)`	*Vero* se gli operandi si equivalgono.
`(< op1 op2…)`	*Vero* se gli operandi sono in ordine crescente.
`(> op1 op2…)`	*Vero* se gli operandi sono in ordine decrescente.
`(<= op1 op2…)`	*Vero* se gli operandi sono in ordine non decrescente.
`(>= op1 op2…)`	*Vero* se gli operandi sono in ordine non crescente.

È interessante notare che le funzioni per il confronto ammettono l'uso di più di due argomenti. Si osservino gli esempi seguenti, con i

risultati che restituiscono:

```
(= 2 2)                 ===> #t
(= 2 2 2)               ===> #t
(= 2 2 2 1)             ===> #f
(< 1 2)                 ===> #t
(< 1 2 3)               ===> #t
(< 1 2 3 2)             ===> #f
```

Tabella u127.36. Elenco delle funzioni per il confronto tra caratteri.

Funzione	Descrizione
`(char=?` ***car1 car2***`)`	*Vero* se i due caratteri sono uguali.
`(char<?` ***car1 car2***`)`	*Vero* se il primo carattere è lessicograficamente inferiore al secondo.
`(char>?` ***car1 car2***`)`	*Vero* se il primo carattere è lessicograficamente superiore al secondo.
`(char<=?` ***car1 car2***`)`	*Vero* se il primo carattere è lessicograficamente non superiore al secondo.
`(char>=?` ***car1 car2***`)`	*Vero* se il primo carattere è lessicograficamente non inferiore al secondo.
`(char-ci=?` ***car1 car2***`)`	Come '**`char=?`**', senza distinguere tra maiuscole e minuscole.
`(char-ci<?` ***car1 car2***`)`	Come '**`char<?`**', senza distinguere tra maiuscole e minuscole.
`(char-ci>?` ***car1 car2***`)`	Come '**`char>?`**', senza distinguere tra maiuscole e minuscole.
`(char-ci<=?` ***car1 car2***`)`	Come '**`char<=?`**', senza distinguere tra maiuscole e minuscole.
`(char-ci>=?` ***car1 car2***`)`	Come '**`char>=?`**', senza distinguere tra maiuscole e minuscole.

Per quanto riguarda il confronto tra caratteri e tra stringhe, non è stabilita la possibilità di inserire più di due argomenti, anche se è possibile che una realizzazione Scheme lo consenta.

```
(char<? #\a #\b)        ===> #t
(char<? #\A #\B)        ===> #t
(char-ci<=? #\A #\b)    ===> #t
(char-ci<=? #\a #\B)    ===> #t
(char-ci=? #\a #\A)     ===> #t
```

Tabella u127.38. Elenco delle funzioni per il confronto tra stringhe.

Funzione	Descrizione
`(string=?` ***str1 str2***`)`	*Vero* se le due stringhe sono uguali.
`(string<?` ***str1 str2***`)`	*Vero* se la prima stringa è lessicograficamente inferiore alla seconda.
`(string>?` ***str1 str2***`)`	*Vero* se la prima stringa è lessicograficamente superiore alla seconda.
`(string<=?` ***str1 str2***`)`	*Vero* se la prima stringa è lessicograficamente non superiore alla seconda.
`(string>=?` ***str1 str2***`)`	*Vero* se la prima stringa è lessicograficamente non inferiore alla seconda.
`(string-ci=?` ***str1 str2***`)`	Come '**`string=?`**', senza distinguere tra maiuscole e minuscole.
`(string-ci<?` ***str1 str2***`)`	Come '**`string<?`**', senza distinguere tra maiuscole e minuscole.
`(string-ci>?` ***str1 str2***`)`	Come '**`string>?`**', senza distinguere tra maiuscole e minuscole.
`(string-ci<=?` ***str1 str2***`)`	Come '**`string<=?`**', senza distinguere tra maiuscole e minuscole.
`(string-ci>=?` ***str1 str2***`)`	Come '**`string>=?`**', senza distinguere tra maiuscole e minuscole.

```
(string<? "ab" "aba")           ===> #t
(string<? "AB" "ABA")           ===> #t
(string-ci<? "AB" "aba")        ===> #t
(string-ci<? "ab" "ABA")        ===> #t
(string-ci=? "ciao" "CIAO")     ===> #t
```

Scheme offre dei predicati particolari per il confronto tra due oggetti, come mostrato nella tabella u127.40. È difficile definire in modo chiaro la differenza che c'è tra questi tre predicati. In generale si può affermare che '**`equal?`**' sia il predicato che è più permissivo, mentre '**`eq?`**' è quello più restrittivo.

Tabella u127.40. Elenco delle funzioni per il confronto tra gli oggetti.

Funzione	Descrizione
`(eq?` ***op1 op2***`)`	*Vero* se i due operandi sono identici.
`(eqv?` ***op1 op2***`)`	*Vero* se i due operandi sono equivalenti dal punto di vista operativo.
`(equal?` ***op1 op2***`)`	*Vero* se i due operandi hanno la stessa struttura e lo stesso contenuto.

```
(equal? "abc" "abc")            ===> #t
(eqv? "abc" "abc")              ===> #f
(eq? "abc" "abc")               ===> #f

(equal? 2 2)                    ===> #t
(eqv? 2 2)                      ===> #t
(eq? 2 2)                       ===> (non specificato)

(equal? 'a 'a)                  ===> #t
(eqv? 'a 'a)                    ===> #t
(eq? 'a 'a)                     ===> #t
```

Caratteri

Alcune funzioni specifiche per i caratteri sono elencate nella tabella u127.42. Per quanto riguarda il caso particolare del predicato '**`char-whitespace?`**', questo si avvera nel caso in cui si tratti di *<SP>*, *<HT>*, *<LF>*, *<FF>* e *<CR>*.

Tabella u127.42. Elenco di alcune funzioni specifiche per la gestione dei caratteri.

Funzione	Descrizione
`(char?` ***oggetto***`)`	*Vero* se l'oggetto è un carattere.
`(char-alphabetic?` ***carattere***`)`	*Vero* se il carattere è alfabetico.
`(char-numeric?` ***carattere***`)`	*Vero* se il carattere è numerico.
`(char-whitespace?` ***carattere***`)`	*Vero* se si tratta di uno spazio orizzontale o verticale.
`(char-upper-case?` ***carattere***`)`	*Vero* se si tratta di un carattere alfabetico maiuscolo.
`(char-lower-case?` ***carattere***`)`	*Vero* se si tratta di un carattere alfabetico minuscolo.
`(char->integer` ***carattere***`)`	Restituisce un numero corrispondente al carattere.
`(integer->char` ***numero_intero***`)`	Restituisce un carattere corrispondente al numero.
`(char-upcase` ***carattere***`)`	Se possibile, converte il carattere in maiuscolo.
`(char-downcase` ***carattere***`)`	Se possibile, converte il carattere in minuscolo.

Nella conversione attraverso le funzioni '**`char->integer`**' e '**`integer->char`**', l'equivalenza tra carattere e numero dipende dalla realizzazione di Scheme; molto probabilmente dipende dalla codifica dell'insieme di caratteri utilizzato.

Stringhe

Alcune funzioni specifiche per i caratteri sono elencate nella tabella u127.43. Quando le funzioni fanno riferimento a un indice per indicare un carattere all'interno di una stringa, si deve ricordare che il primo corrisponde alla posizione zero. Quando si fa riferimento a due indici, uno per indicare il carattere iniziale e uno per fare riferimento al carattere finale, il secondo indice deve puntare alla posizione successiva all'ultimo carattere da prendere in considerazione. Questo permette di individuare una stringa nulla quando l'indice iniziale e l'indice finale sono uguali.

Tabella u127.43. Elenco di alcune funzioni specifiche per la gestione delle stringhe.

Funzione	Descrizione
`(string?` ***oggetto***`)`	*Vero* se l'oggetto è una stringa.

Funzione	Descrizione
`(make-string` ***numero_caratteri*** `)`	Restituisce una stringa della lunghezza indicata.
`(make-string` ***numero_caratteri*** `→` ***carattere*** `)`	Restituisce una stringa composta con il carattere indicato.
`(string` ***carattere…*** `)`	Restituisce una stringa composta dai caratteri indicati.
`(string-length` ***stringa*** `)`	Restituisce il numero di caratteri contenuto.
`(string-ref` ***stringa indice*** `)`	Restituisce il carattere nella posizione dell'indice.
`(string-set!` ***stringa indice carattere*** `)`	Modifica il carattere che si trova nella posizione dell'indice.
`(substring` ***stringa inizio fine*** `)`	Estrae la sottostringa compresa tra i due indici.
`(string-append` ***stringa…*** `)`	Restituisce una stringa unica complessiva.
`(string-copy` ***stringa*** `)`	Restituisce una copia della stringa.
`(string-fill!` ***stringa carattere*** `)`	Sostituisce gli elementi della stringa con il carattere indicato.
`(string->list` ***stringa*** `)`	Restituisce una lista composta dai caratteri della stringa.
`(list->string` ***lista_di_caratteri*** `)`	Restituisce una stringa a partire da una lista di caratteri.

```
(make-string 10 #\A)            ===> "AAAAAAAAAA"
(string-length "ciao")          ===> 4

(define a "ciao")
(string-set! a 0 #\C)
a                               ===> "Ciao"
(substring a 2 4)               ===> "ao"
```

Strutture di controllo

Anche con Scheme sono disponibili le strutture di controllo comuni nei linguaggi di programmazione. Evidentemente, queste sono realizzate attraverso delle funzioni. In base a tale impostazione, per sottoporre una parte di codice alla verifica di una condizione, o per metterla in un ciclo, occorre che questa sia inserita in una funzione che possa essere chiamata all'interno di un'espressione.

Per intendere il problema, si osservi l'esempio seguente, che mostra la scelta tra la chiamata della funzione '**`display`**' per visualizzare il messaggio «bello», o «brutto», in funzione di una condizione (che in questo caso si avvera necessariamente):

```
(if (> 3 2) (display "bello") (display "brutto"))
```

Per ovviare a questo inconveniente si può utilizzare la funzione '**`begin`**', che permette di incorporare più espressioni dove invece se ne potrebbe inserire una sola.

Funzione «begin»

Per tutte le situazioni in cui è possibile indicare una sola espressione, mentre invece se ne vorrebbero inserire diverse, esiste la funzione '**`begin`**':

```
(begin
    espressione
    espressione
    …
)
```

Il senso si comprende intuitivamente: le espressioni che costituiscono gli argomenti di '**`begin`**' vengono valutate in ordine, da sinistra a destra (in questo caso dall'alto in basso). L'esempio seguente è molto banale: visualizza un messaggio e termina.


```
(begin
    (display "ciao ")
    (display "a ")
    (display "tutti!")
    (newline)
)
```

È importante osservare che all'interno della funzione '**`begin`**' non è possibile dichiarare delle variabili locali, a meno che per questo si inseriscano delle altre funzioni che creano un loro ambiente, come '**`let`**' e le altre simili.

Struttura condizionale: «if»

La struttura condizionale è il sistema di controllo fondamentale dell'andamento del flusso delle istruzioni.

`(if` ***condizione espressione_se_vero*** `[`***espressione_se_falso***`])`

La funzione '**`if`**' valuta i suoi argomenti in un ordine preciso: per prima cosa viene valutato il primo argomento; se il risultato è *Vero*, o comunque se si ottiene un risultato equiparabile a *Vero*, valuta il secondo argomento; in alternativa, valuta il terzo argomento, se è stato fornito. Alla fine restituisce il valore dell'ultima espressione a essere stata valutata (ammesso che questa restituisca qualcosa). Sotto vengono mostrati alcuni esempi in cui alcune parti del programma sono state saltate per non distrarre l'attenzione del lettore:

```
(define Importo 0)
…
(if (> Importo 10000000) (display "L'offerta è vantaggiosa"))
```

```
(define Importo 0)
…
(if (> Importo 10000000)
    (display "L'offerta è vantaggiosa")
    (display "Lascia perdere")
)
```

```
(define Importo 0)
…
(if (> Importo 10000000)
    (display "L'offerta è vantaggiosa")
    (if (> Importo 5000000)
        (display "L'offerta è accettabile")
        (display "Lascia perdere")
    )
)
```

Come accennato, potrebbe essere conveniente l'utilizzo della funzione '**`begin`**' per facilitare la descrizione di gruppi di istruzioni (espressioni). Si osservi l'esempio seguente, in cui viene salvato il valore dell'importo nella variabile '**`Offerta`**':

```
(define Importo 0)
(define Offerta 0)
…
(if (> Importo 10000000)
    ; then
    (begin
        (display "L'offerta è vantaggiosa")
        (set! Offerta Importo)
        ; eventualmente fa anche qualcosa in più
        ;…
    )
    ; else
    (if (> Importo 5000000)
        ; then
        (begin
            (display "L'offerta è accettabile")
            (set! Offerta Importo)
            ; eventualmente fa anche qualcosa in più
            ;…
        )
        ; else
        (display "Lascia perdere")
    ; end if
    )
; end if
)
```

Struttura di selezione: «cond»

Scheme fornisce due strutture di selezione. In questo caso, la funzione '**`cond`**' si basa sulla verifica di condizioni distinte per ogni blocco di espressioni.

```
(cond
    (condizione  espressione…)
    (condizione  espressione…)
    …
    [(else espressione…)]
)
```

Lo schema sintattico dovrebbe essere chiaro a sufficienza: la funzione '**cond**' ha come argomenti una serie di «blocchi» (si tratta di liste, ma questo viene chiarito quando si passa alla descrizione delle liste), contenenti ognuno un'espressione iniziale che deve essere valutata per determinare se le espressioni successive devono essere valutate o meno. Nel momento in cui si incontra una condizione che si avvera, i blocchi successivi vengono ignorati. Se non si incontra alcuna condizione che si avvera, se esiste l'ultimo blocco, corrispondente alla funzione '**else**', le espressioni relative vengono eseguite.

A differenza della funzione '**if**', in questo caso si possono indicare più espressioni per ogni condizione della selezione; in questo senso, la funzione '**cond**' può diventare un sostituto opportuno di quella. Segue un esempio tipico di selezione:

```
(define Mese 0)
…
(cond
    ((= Mese 1) (display "gennaio") (newline))
    ((= Mese 2) (display "febbraio") (newline))
    ((= Mese 3) (display "marzo") (newline))
    ((= Mese 4) (display "aprile") (newline))
    ((= Mese 5) (display "maggio") (newline))
    ((= Mese 6) (display "giugno") (newline))
    ((= Mese 7) (display "luglio") (newline))
    ((= Mese 8) (display "agosto") (newline))
    ((= Mese 9) (display "settembre") (newline))
    ((= Mese 10) (display "ottobre") (newline))
    ((= Mese 11) (display "novembre") (newline))
    ((= Mese 12) (display "dicembre") (newline))
    (else (display "mese errato!") (newline))
)
```

Struttura di selezione: «case»

Scheme fornisce anche la struttura di selezione tradizionale, ovvero la funzione '**case**', che si basa sulla verifica del valore di una sola «chiave». Anche '**case**' permette l'indicazione di più espressioni per ogni elemento della selezione.

```
(case espressione_di_selezione
    ((dato…)  espressione…)
    ((dato…)  espressione…)
    …
    [(else espressione…)]
)
```

La prima espressione a essere valutata è quella che costituisce il primo argomento della funzione '**case**'. Successivamente, il suo risultato viene comparato con quello dei «dati» elencati all'inizio di ogni gruppo di espressioni (si vedano gli esempi). Se la comparazione ha successo, allora vengono valutate le espressioni successive (all'interno del blocco), nell'ordine in cui si trovano. Se il confronto non ha successo, se esiste un blocco finale costituito dalla funzione '**else**', vengono eseguite le espressioni relative. Seguono alcuni esempi:

```
(define Mese 0)
…
(case Mese
    ((1) (display "gennaio") (newline))
    ((2) (display "febbraio") (newline))
    ((3) (display "marzo") (newline))
    ((4) (display "aprile") (newline))
    ((5) (display "maggio") (newline))
    ((6) (display "giugno") (newline))
    ((7) (display "luglio") (newline))
    ((8) (display "agosto") (newline))
    ((9) (display "settembre") (newline))
    ((10) (display "ottobre") (newline))
    ((11) (display "novembre") (newline))
    ((12) (display "dicembre") (newline))
    (else (display "mese errato!") (newline))
)
```



```
(define Anno 0)
(define Mese 0)
(define Giorni 0)
…
(case Mese
    ((1 3 5 7 8 10 12) (set! Giorni 31))
    ((4 6 9 11) (set! Giorni 30))
    ((2)
        (if
            (or
                (and (= (modulo Anno 4) 0) (not (= (modulo Anno 100) 0)))
                (= (modulo Anno 400) 0)
            )
            (set! Giorni 29)
            (set! Giorni 28)
        )
    )
)
```

Iterazione: «do»

Scheme dispone di una funzione unica per realizzare i cicli iterativi e quelli enumerativi. Si tratta di '**do**', il cui funzionamento è, a prima vista, un po' strano. Come ciclo iterativo la sintassi si riduce al modello seguente:

```
(do ()
    (condizione_di_uscita [espressione_pre_uscita…])
    espressione_del_ciclo…
)
```

In questa forma, viene valutata prima la condizione; se si avvera, vengono valutate le espressioni successive, quelle contenute nello spazio delle parentesi (la lista della condizione), quindi il ciclo termina. Se la condizione non si avvera, vengono eseguite le espressioni esterne al blocco della condizione, al termine delle quali riprende il ciclo.

Quando si vuole usare la funzione '**do**' per realizzare un ciclo enumerativo, si definiscono una o più variabili da inizializzare e modificare in qualche modo a ogni ciclo:

```
(do ((variabile  inizializzazione  passo)…)
    (condizione_di_uscita [espressione_pre_uscita…])
    espressione_del_ciclo…
)
```

Le variabili vengono dichiarate (allocate) dalla funzione '**do**' stessa, avendo effetto solo in ambito locale, all'interno della funzione che le dichiara (in pratica, mascherano temporaneamente altre variabili esterne con lo stesso nome). Le variabili vengono inizializzate immediatamente con il valore ottenuto dall'espressione di inizializzazione, quindi inizia il primo ciclo. Alla fine di ogni ciclo, prima dell'inizio del successivo, vengono valutate le espressioni del passo, assegnando alle variabili relative i valori che si ottengono.

L'esempio seguente fa apparire per 10 volte la lettera «x». Si osservi l'uso di una variabile esterna per scandire i cicli:

```
(define Contatore 0)

(do () ((>= Contatore 10))
    ; incrementa il contatore di un'unità
    (set! Contatore (+ Contatore 1))
    (display "x")
)

(newline)
```

La stessa cosa avrebbe potuto essere ottenuta dichiarando la variabile all'interno della funzione '**do**':

```
(do ((Contatore 0 Contatore))
    ; condizione di uscita
    ((>= Contatore 10))
    ; incrementa il contatore di un'unità
    (set! Contatore (+ Contatore 1))
    (display "x")
)

(newline)
```

Infine, si può trasferire l'incremento del contatore nel blocco in cui si dichiara e si inizializza la variabile '**Contatore**':

```
(do ((Contatore 0 (+ Contatore 1)))
    ; condizione di uscita
    ((>= Contatore 10))
    ; istruzioni del ciclo
    (display "x")
)

(newline)
```

Conclusione di un programma Scheme

Un programma Scheme termina quando si esauriscono le istruzioni, oppure quando viene incontrata e valutata la funzione '**exit**'.

```
(exit [valore_di_uscita])
```

Come si vede dallo schema sintattico, è possibile indicare un numero che si traduce poi nel valore di uscita del programma stesso.

L'utilizzo di questa funzione all'interno di un ambiente di interpretazione Scheme, serve normalmente a concludere il funzionamento del programma relativo.

Riferimenti

- A. Aaby, *Scheme Tutorial*, 1996
  http://cs.wwc.edu/~cs_dept/KU/PR/Scheme.html
- Pierre Castéran, Robert Cori, *Passeport pour Scheme*
  Il documento citato sembra essere scomparso dalla rete, probabilmente in vista di una sua pubblicazione. In origine, si trovava presso *http://dept-info.labri.u-bordeaux.fr/~cori/Bouquins/scheme.ps* .
- *R^{5}RS -- Revised-5 Report on the Algorithmic Language Scheme*, 1998
  http://www.swiss.ai.mit.edu/~jaffer/r5rs_toc.html
  http://www.swiss.ai.mit.edu/ftpdir/scheme-reports/r5rs.ps.gz

Scheme: struttura del programma e campo di azione

Definizione e campo di azione 1063
Ridefinizione 1064
Definizione «lambda» 1065
Ricorsione 1066
Funzioni «let», «let*» e «letrec» 1066

Nel capitolo introduttivo, sono state elencate le strutture elementari per il controllo e il raggruppamento delle istruzioni (espressioni) di Scheme. In questo capitolo, si vuole mostrare in che modo possano essere definite delle funzioni, o comunque dei raggruppamenti di istruzioni all'interno dei quali si possa individuare un campo di azione locale per le variabili che vi vengono dichiarate.

Le funzioni del linguaggio Scheme prevedono il passaggio di parametri solo **per valore**; questo significa che gli argomenti di una funzione vengono valutati prima di tutto. Al posto del passaggio dei parametri per riferimento, Scheme consente l'indicazione di espressioni costanti, concetto a cui si è accennato nel capitolo precedente.

Definizione e campo di azione

La definizione e inizializzazione di un oggetto avviene normalmente attraverso la funzione '**define**'. Questa può servire per dichiarare una variabile normale, o anche per dichiarare una funzione.

```
(define nome_variabile espressione_di_inizializzazione)
```

Quello che si vede sopra è appunto lo schema sintattico per la dichiarazione e inizializzazione di una variabile, cosa che è stata vista più volte nel capitolo precedentemente. Sotto, si vede lo schema sintattico per la dichiarazione di una funzione:

```
(define (nome_funzione elenco_parametri_formali)
    corpo
)
```

In questo caso, i parametri formali sono dei nomi che rappresentano i parametri della funzione che viene dichiarata, mentre il corpo è costituito da una serie di espressioni, che rappresentano il contenuto della funzione che si dichiara. Il valore che viene restituito dall'ultima espressione che viene eseguita all'interno della funzione, è ciò che restituisce la funzione stessa. L'esempio seguente, serve a definire la funzione '**moltiplica**' con due parametri, '**x**' e '**y**', che restituisce il prodotto dei suoi due argomenti:

```
(define (moltiplica x y)
    ; il corpo di questa funzione è molto breve
    (* x y)
)
```

Per chiamare questa funzione, basta semplicemente un'istruzione come quella seguente:

```
(moltiplica 10 11)              ===> 110
```

Le dichiarazioni di questo tipo, cioè di variabili e di funzioni, possono avvenire solo nella parte più esterna di un programma Scheme, oppure all'interno della dichiarazione di altre funzioni e delle altre strutture descritte in questo capitolo, ma in tal caso devono apparire all'inizio del «corpo» delle espressioni che queste strutture contengono. Si osservi l'esempio seguente, in cui viene dichiarata una funzione e al suo interno si dichiarano altre variabili locali:

```
(define (moltiplica x y)
    ; dichiara le variabili locali
    (define z 0)

    ; definisce un ciclo enumerativo, per il calcolo del prodotto
    ; attraverso la somma, in cui viene dichiarata implicitamente
    ; la variabile «i».
    (do ((i 1 (+ i 1)))
        ; condizione di uscita
        ((> i y))
        ; istruzioni del ciclo
        (set! z (+ z x))
    )
    ; al termine restituisce il valore contenuto nella variabile «z»
    z
)
```

Dovrebbe essere intuitivo, quindi, che il campo di azione delle variabili dichiarate all'interno di una funzione '**`define`**' è limitato alla funzione stessa. La stessa cosa varrebbe per le funzioni, dichiarate all'interno di un ambiente del genere. Si osservi l'esempio seguente, in cui si calcola il prodotto tra due numeri, a partire dalla somma di questi, ma dove la somma si ottiene da un'altra funzione, locale, che a sua volta la calcola con incrementi di una sola unità alla volta.

```
(define (moltiplica x y)
    ; dichiara la funzione «somma», locale nell'ambito della
    ; funzione «moltiplica»
    (define (somma x y)
        ; dichiara una variabile locale per la funzione «somma»,
        ; che comunque non serve a nulla :-)
        (define z 2000)
        ; definisce un ciclo enumerativo, per il calcolo della
        ; somma, sommando un'unità alla volta
        (do ()
            ; condizione di uscita
            ((<= y 0))
            ; istruzioni del ciclo
            (set! x (+ x 1))
            ; decrementa «y»
            (set! y (- y 1))
        )
        ; al termine restituisce il valore contenuto nella variabile «x»
        x
    ; fine della funzione locale «somma»
    )
    ; dichiara le variabili locali della funzione «moltiplica»
    (define z 0)
    ; definisce un ciclo enumerativo, per il calcolo del prodotto
    ; attraverso la somma, in cui viene dichiarata implicitamente
    ; la variabile «i».
    (do ((i 1 (+ i 1)))
        ; condizione di uscita
        ((> i y))
        ; istruzioni del ciclo
        (set! z (somma z x))
    )
    ; al termine restituisce il valore contenuto nella variabile «z»
    z
)
```

Questo esempio è solo un pretesto per mostrare che le variabili locali '**`x`**', '**`y`**' e '**`z`**', della funzione '**`somma`**' hanno effetto solo nell'ambito di questa funzione; inoltre, la funzione '**`somma`**' e le variabili locali '**`x`**', '**`y`**' e '**`z`**', della funzione '**`moltiplica`**', hanno effetto solo nell'ambito della funzione '**`moltiplica`**' stessa.

Ridefinizione

«

Nel capitolo introduttivo si è accennato al fatto che la ridefinizione di una variabile, o di una funzione, implica una nuova allocazione di memoria, senza liberare quella utilizzata precedentemente. Pertanto, i riferimenti fatti in precedenza a quell'oggetto, continuano a utilizzare in pratica la vecchia allocazione. Si osservi l'esempio seguente:

```
(define x 20)
x                       ===> 20
(define y (* 2 x))
y                       ===> 40
(define x 100)
x                       ===> 100
y                       ===> 40
```

Quanto mostrato con questo esempio, non ha nulla di eccezionale, rispetto ai linguaggi di programmazione tradizionali. Tuttavia, potrebbe risultare strano da un punto di vista strettamente matematico. Se invece lo scopo fosse quello di definire un sistema di equazioni, '**`y`**' dovrebbe essere trasformato in una funzione, come nell'esempio seguente:


```
(define x 20)
x                       ===> 20
(define (f y) (* 2 x))
(f)                     ===> 40
(define x 100)
x                       ===> 100
(f)                     ===> 200
```

Qualunque oggetto con un identificatore può essere ridefinito. Si osservi l'esempio seguente, in cui si imbrogliano le carte e si fa in modo che l'identificatore '**`*`**' corrisponda a una funzione che esegue la somma, mentre prima valeva per una moltiplicazione:

```
(define (* x y) (+ x y)
(* 3 5)                 ===> 8
```

Si ricorda che per modificare il contenuto di una variabile allocata, senza allocare un'altra area di memoria, si utilizza generalmente la funzione '**`set!`**'.

Definizione «lambda»

«

Scheme tratta gli identificatori delle funzioni (i loro nomi), nello stesso modo di quelli delle variabili. In altri termini, le funzioni sono variabili che contengono un riferimento a un blocco di codice. È possibile dichiarare una funzione attraverso la funzione '**`lambda`**', che restituisce la funzione stessa. In questo modo, una funzione può essere dichiarata anche attraverso l'assegnamento di una variabile, che poi diventa una funzione a tutti gli effetti.

Prima di vedere come si usa la dichiarazione di una funzione attraverso la funzione '**`lambda`**', è bene ribadire che, attraverso questo meccanismo, è possibile dichiarare una funzione in tutte quelle situazioni in cui è possibile inizializzare o assegnare una variabile.

```
(lambda (elenco_parametri_formali)
    corpo
)
```

Come si vede dal modello sintattico, la funzione '**`lambda`**' è relativamente semplice: il primo argomento è un blocco contenente l'elenco dei nomi (locali) dei parametri formali; gli argomenti successivi sono le espressioni che costituiscono il corpo della funzione. Non si dichiara il nome della funzione, dal momento che '**`lambda`**' restituisce la funzione stessa, che viene poi identificata (ammesso che lo si voglia fare) dalla variabile a cui questa viene assegnata.

All'inizio del «corpo» delle espressioni che descrivono il contenuto della funzione che si dichiara, si possono inserire delle dichiarazioni ulteriori attraverso la funzione '**`define`**'.

Sotto vengono proposti alcuni esempi che dovrebbero lasciare intendere in quante situazioni si può utilizzare una dichiarazione di funzione attraverso '**`lambda`**'.

```
; dichiara la variabile «f» e la inizializza temporaneamente al valore zero
(define f 0)
; assegna a «f» una funzione che esegue la somma dei suoi due argomenti
(set! f
    (lambda (x y)
        (+ x y)
    )
)
; calcola la somma tra 4 e 5, restituendo 9
(f 4 5)
```

L'esempio che appare sopra mostra in che modo si possa dichiarare una funzione in qualunque situazione in cui si può assegnare un valore a una variabile.

```
; dichiara direttamente la funzione «f»
(define f
    ; inizializza «f» con una funzione che esegue la somma
    ; dei suoi due argomenti
    (lambda (x y)
        ; corpo della dichiarazione della funzione
        (+ x y)
    )
)
; calcola la somma tra 4 e 5, restituendo 9
(f 4 5)
```

In questo caso, l'assegnamento della funzione alla variabile '**`f`**' è avvenuto contestualmente alla dichiarazione della variabile stessa.

```
(define moltiplica
    (lambda (x y)
        ; dichiara le variabili locali
        (define z 0)
        ; definisce un ciclo enumerativo, per il calcolo del prodotto
        ; attraverso la somma, in cui viene dichiarata implicitamente
        ; la variabile «i».
        (do ((i 1 (+ i 1)))
            ; condizione di uscita
            ((> i y))
            ; istruzioni del ciclo
            (set! z (+ z x))
        )
        ; al termine restituisce il valore contenuto nella variabile «z»
        z
    )
)
```

Questo esempio, mostra in che modo possano avvenire delle dichiarazioni locali nel corpo di una dichiarazione '**`lambda`**'.

L'esempio successivo è un po' un estremo, nel senso che viene mostrata la dichiarazione di una funzione «anonima», che viene usata immediatamente per calcolare il prodotto tra tre e quattro. Successivamente al suo utilizzo istantaneo, non c'è modo di riutilizzare tale funzione.

```
(
    ; dichiarazione della funzione anonima
    (lambda (x y)
        ; dichiara le variabili locali
        (define z 0)
        ; definisce un ciclo enumerativo, per il calcolo del prodotto
        ; attraverso la somma, in cui viene dichiarata implicitamente
        ; la variabile «i».
        (do ((i 1 (+ i 1)))
            ; condizione di uscita
            ((> i y))
            ; istruzioni del ciclo
            (set! z (+ z x))
        )
        ; al termine restituisce il valore contenuto nella variabile «z»
        z
    )
    ; indicazione del primo argomento
    3
    ; indicazione del secondo argomento
    4
)
```

Ricorsione

Si intuisce la possibilità di Scheme di scrivere funzioni ricorsive. Non dovrebbe essere difficile arrivare a questo risultato senza spiegazioni particolari. L'esempio seguente mostra il calcolo del fattoriale attraverso una funzione ricorsiva:

```
(define (fattoriale n)
    (if (= n 0)
        ; then
        1
        ; else
        (* n (fattoriale (- n 1)))
    )
)
```

Si intuisce che una funzione senza nome, come nel caso di quella dichiarata con '**`lambda`**', senza assegnarla a una variabile, non può essere resa ricorsiva, a meno di definire una sotto-funzione ricorsiva al suo interno. L'esempio seguente è una variante di quello precedente, in cui viene utilizzata una dichiarazione '**`lambda`**'.

```
(define fattoriale
    (lambda (n)
        (if (= n 0)
            ; then
            1
            ; else
            (* n (fattoriale (- n 1)))
        )
    )
)
```

Funzioni «let», «let*» e «letrec»

Le funzioni '**`let`**', '**`let*`**' e '**`letrec`**', hanno lo scopo di circoscrivere un ambiente, all'interno del quale può essere inserita una serie indefinita di espressioni (istruzioni), prima delle quali vengono dichiarate delle variabili il cui campo di azione è locale rispetto a quell'ambito.


```
(let ((variabile inizializzazione)…)
    corpo
)
```

```
(let* ((variabile inizializzazione)…)
    corpo
)
```

```
(letrec ((variabile inizializzazione)…)
    corpo
)
```

In tutti e tre le forme, le variabili vengono inizializzate e quindi si passa alla valutazione delle espressioni successive (le istruzioni). Alla fine, la funzione restituisce il valore dell'ultima espressione a essere stata eseguita al suo interno.

Nel caso di '**`let`**', le variabili vengono dichiarate e inizializzate senza un ordine preciso, ma semplicemente prima di passare alla valutazione delle espressioni successive:

```
(let ((x 1) (y 2))
    (+ x y)
)                       ===> 3
```

L'esempio non ha un grande significato da un punto di vista pratico, ma si limita a mostrare intuitivamente come si comporta la funzione '**`let`**'. In questo caso, vengono dichiarate le variabili locali '**`x`**' e '**`y`**', inizializzandole rispettivamente a uno e due, infine viene calcolata semplicemente la somma tra le due variabili, cosa che restituisce il valore tre.

Nel caso di '**`let*`**', le variabili vengono dichiarate e inizializzate nell'ordine in cui sono (da sinistra a destra); pertanto, ogni inizializzazione può fare riferimento alle variabili dichiarate precedentemente nella stessa sequenza:

```
(let* ((x 1) (y (+ x 1)))
    (+ x y)
)                       ===> 3
```

L'esempio mostra che la variabile locale '**`y`**' viene inizializzata partendo dal valore della variabile locale '**`x`**', incrementando il valore di questa di un'unità.

La funzione '**`letrec`**' è più sofisticata; il nome sta per *let recursive*. È un po' difficile spiegare il senso di questa; si tenta almeno di mostrare la cosa in modo intuitivo.

Nello stesso modo in cui si può dichiarare una variabile, si può dichiarare una funzione. In questo senso, tali dichiarazioni possono anche essere ricorsive all'interno di una funzione '**`letrec`**'. Viene mostrato un esempio tratto da *R*^{5}*RS*:

```
(letrec
    ; dichiara le «variabili», che in realtà sono funzioni (predicati)
    (
        ; dichiara la funzione «pari?»
        (pari?
            (lambda (n)
                (if (zero? n)
                    ; il numero è pari
                    #t
                    ; altrimenti si prova a vedere se è dispari
                    (dispari? (- n 1))
                )
            )
        )
        ; dichiara la funzione «dispari?»
        (dispari?
            (lambda (n)
                (if (zero? n)
                    ; il numero è dispari
                    #f
                    ; altrimenti si prova a vedere se è pari
                    (pari? (- n 1))
                )
            )
        )
    )
    ; fine della dichiarazione delle variabili

    ; verifica che il numero 88 è pari, chiamando la funzione
    ; «pari?» dichiarata all'inizio
    (pari? 88)
    ; la chiamata restituisce il valore #t e, di conseguenza,
```

```
    ; è questo il valore restituiti da tutto
)
```

Le variabili '**pari?**' e '**dispari?**' vengono inizializzate assegnando loro una funzione dichiarata con '**lambda**' e il loro scopo è quello di verificare che l'argomento sia rispettivamente un numero pari o dispari.

```
(pari? 2)               ===> #t
(dispari? 2)            ===> #f
```

Tali variabili e di conseguenza queste funzioni, hanno effetto solo nell'ambito della dichiarazione '**letrec**', al termine della quale diventano semplicemente irraggiungibili. Il principio di funzionamento di queste funzioni, sta nel fatto che lo zero sia pari, di conseguenza:

```
(pari? 0)               ===> #t
(dispari? 0)            ===> #f
```

Per tutti i numeri superiori, invece, è sufficiente verificare in modo ricorsivo di che tipo è il valore $n-1$. Per la precisione, se si sta verificando il fatto che un numero sia pari, se questo è superiore a zero, si può verificare che quel numero, meno uno, sia dispari, continuando così, di seguito.

Queste tre strutture sono importanti soprattutto perché consentono di inserire delle dichiarazioni di variabili o di funzioni, oltre al fatto che così circoscrivono un ambito locale per queste. Come si è visto, queste dichiarazioni possono essere fatte anche prima (anche con '**let**' e '**let***'), tenendo conto dell'ordine di valutazione che ognuna di queste strutture garantisce.

```
(let ((x 1) (y 2))
    (define messaggio "sto calcolando la somma...")
    (display messaggio)
    (newline)
    (+ x y)
)                       ===> 3
```

L'esempio che si vede sopra, è solo un'estensione di quanto già visto sopra, allo scopo di mostrare la possibilità di utilizzare la funzione '**define**' all'inizio del corpo di espressioni che contiene. L'esempio successivo è una variante ulteriore, in cui il messaggio viene dichiarato tra le variabili iniziali di '**let**'.

```
(let ((x 1) (y 2) (messaggio "sto calcolando la somma..."))
    (display messaggio)
    (newline)
    (+ x y)
)                       ===> 3
```


Scheme: liste e vettori

Liste e coppie 1069
- Dichiarazione di una lista 1070
- Caratteristiche esteriori di una lista 1070
- Operazioni fondamentali con le liste 1070
- Funzioni tipiche sulle liste 1072

Vettori 1073

Strutture di controllo applicate alle liste 1073
- Funzione «apply» 1074
- Funzione «map» 1074
- Funzione «for-each» 1074

Riferimenti 1074

Scheme dispone di due strutture di dati particolari: liste e vettori. Le liste sono una sequenza di elementi a cui si accede con una certa difficoltà, senza la possibilità di utilizzare un indice, mentre i vettori sono l'equivalente degli array degli altri linguaggi.

Liste e coppie

La lista è la struttura di dati fondamentale di Scheme. In questo linguaggio, le stesse istruzioni (le chiamate delle funzioni) sono espresse in forma di lista:

```
(elemento...)
```

La lista è un elenco di elementi ordinati. Gli elementi di una lista possono essere oggetti di qualunque tipo, comprese altre liste. Ci sono molte situazioni in cui i parametri di una funzione di Scheme sono delle liste; per esempio la dichiarazione di una funzione, attraverso '**define**':

```
(define (nome_funzione elenco_parametri_formali)
    corpo
)
```

Come si vede, il primo parametro della funzione '**define**' è una lista, in cui il primo elemento è il nome della funzione che si crea, mentre gli elementi successivi sono la descrizione dei parametri formali.

Le liste vuote, sono rappresentate da una coppia di parentesi aperta e chiusa, '**()**', rappresentando degli oggetti speciali nella filosofia di Scheme.

Tabella u129.1. Elenco di alcune funzioni specifiche per la gestione delle stringhe.

Funzione	Descrizione
`(list?` *oggetto*`)`	*Vero* se l'oggetto è una lista.
`(pair?` *oggetto*`)`	*Vero* se l'oggetto è una coppia (una lista non vuota).
`(null?` *lista*`)`	*Vero* se la lista è vuota.
`(length` *lista*`)`	Restituisce il numero di elementi della lista.
`(car` *lista*`)`	Restituisce il primo elemento di una lista.
`(cdr` *lista*`)`	Restituisce una lista da cui è stato tolto il primo elemento.
`(cadr` *lista*`)`	`(car (cdr` *lista*`))`
`(cddr` *lista*`)`	`(cdr (cdr` *lista*`))`
`(caadr` *lista*`)`	`(car (car (cdr` *lista*`)))`
`(caddr` *lista*`)`	`(car (cdr (cdr` *lista*`)))`

Funzione	Descrizione
`(cons` ***elemento lista***`)`	Restituisce una lista in cui inserisce al primo posto l'elemento indicato.
`(list` ***elemento…***`)`	Restituisce una lista composta dagli elementi indicati.
`(append` ***lista lista***`)`	Restituisce una lista composta dagli elementi delle due liste indicate.
`(reverse` ***lista***`)`	Restituisce una lista con gli elementi in ordine inverso.
`(set-car!` ***lista oggetto***`)`	Memorizza nella prima posizione della lista l'oggetto indicato.
`(set-cdr!` ***lista oggetto***`)`	Memorizza nella parte successiva al primo elemento l'oggetto indicato.
`(list-tail` ***lista k***`)`	Restituisce una lista in cui mancano i primi ***k*** elementi.
`(list-ref` ***lista k***`)`	Restituisce l'elemento (***k*** + 1)-esimo della lista.
`(vector->list` ***vettore***`)`	Converte il vettore in lista.
`(list->vector` ***list***`)`	Converte la lista in vettore.

Dichiarazione di una lista

«

La dichiarazione di una lista avviene nello stesso modo in cui si dichiara una variabile normale:

```
(define variabile lista_costante)
```

Tuttavia, occorre tenere presente che una lista può essere interpretata come la chiamata di una funzione e come tale verrebbe intesa in questa situazione. Per evitare che ciò avvenga, la si indica attraverso un'espressione costante, cioè la si fa precedere da un apostrofo, o la si inserisce in una funzione '**quote**'. L'esempio seguente dichiara la lista '**lis**' composta dall'elenco dei numeri interi da uno a sei:

```
(define lis '(1 2 3 4 5 6))
```

In questo caso, se la lista non venisse indicata con l'apostrofo, si otterrebbe la valutazione della lista stessa, prima dell'inizializzazione della variabile '**lis**', provocando un errore, dal momento che l'oggetto '**1**' (uno) non esiste.

Caratteristiche esteriori di una lista

«

Le caratteristiche esteriori di una lista sono semplicemente la lunghezza, espressa in numero di elementi, e il fatto che contengano o meno qualcosa. Per verificare queste caratteristiche sono disponibili due funzioni, '**null?**' e '**length**', che richiedono come argomento una lista. Si osservino gli esempi seguenti.

```
; dichiara la lista «lis»
(define lis (1 2 3 4 5 6)

; verifica se la lista «lis» è vuota
(null? lis)                                ===> #f

; calcola la lunghezza della lista
(length lis)                               ===> 6
```

Se fosse stata fornita la lista in modo letterale, senza la variabile '**lis**', la stessa cosa avrebbe dovuto essere scritta nel modo seguente:

```
; verifica se la lista è vuota
(null? '(1 2 3 4 5 6))                     ===> #f

; calcola la lunghezza della lista
(length '(1 2 3 4 5 6))                    ===> 6
```

Operazioni fondamentali con le liste

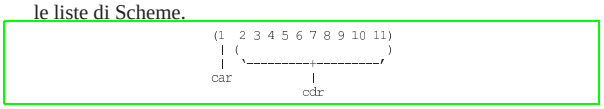
«

L'accesso agli elementi singoli di una lista è un'impresa piuttosto complessa che si attua fondamentalmente con le funzioni '**car**' e '**cdr**'. A queste due si affianca anche '**cons**', il cui scopo è quello di «costruire» una lista.

Per comprendere il senso di queste funzioni, occorre tenere presente che per Scheme una lista è una ***coppia*** composta dal primo elemento, ovvero l'elemento '**car**', e dalla parte restante, ovvero la parte '**cdr**'.

Per la precisione, una coppia è una lista, mentre la lista vuota non è una coppia. La lista contenente un solo elemento, è la composizione dell'unico elemento a disposizione e della lista vuota.

Figura u129.5. La parte «car» e la parte «cdr» che compongono le liste di Scheme.




```
(car lista)
```

```
(cdr lista)
```

Le due funzioni '**car**' e '**cdr**' hanno come argomento una lista, della quale restituiscono, rispettivamente, il primo elemento e la lista restante quando si elimina il primo elemento. Si osservino gli esempi seguenti.[1]

```
(car '(1 2 3 4 5 6))           ===> 1
(cdr '(1 2 3 4 5 6))           ===> (2 3 4 5 6)
```

Data l'idea che ha Scheme sulle liste, la funzione '**cons**' crea una lista a partire dalle sue parti '**car**' e '**cdr**':

```
(cons elemento_car lista_cdr)
```

In altri termini, '**cons**' aggiunge un elemento all'inizio della lista indicata come secondo argomento. Si osservi l'esempio.

```
(cons 0 '(1 2 3 4 5 6))        ===> (0 1 2 3 4 5 6)
```

Le tre funzioni '**car**', '**cdr**' e '**cons**' si completano a vicenda, in base alla relazione schematizzata dalla figura u129.9.

Se viene fornita una lista come primo argomento della funzione '**car**', questa viene inserita come primo elemento della lista risultante.

```
(cons '(0 1 2) '(1 2 3 4 5 6))         ===> ((0 1 2) 1 2 3 4 5 6)
```

Figura u129.9. Relazione che lega le funzioni '**car**', '**cdr**' e '**cons**'. In particolare, «x» e «y» sono liste non vuote; «a» è un elemento ipotetico di una lista.

```
(cons (car x) (cdr x)) = x
(car (cons a y)) = a
(cdr (cons a y)) = y
```

Altri modi per creare una lista sono dati dalle funzioni '**list**' e '**append**'.

```
(list elemento…)
```

```
(append lista lista)
```

La funzione '**list**' restituisce una lista composta dai suoi argomenti (se non si vuole che questi siano valutati prima, occorre ricordare di usare l'apostrofo); la funzione '**append**' restituisce una lista composta dagli elementi delle due liste indicate come argomento (se le liste vengono fornite in modo letterale, occorre ricordare di usare l'apostrofo, per evitare che vengano valutate come funzioni).

```
(list 1 2 3 4 5 6)             ===> (1 2 3 4 5 6)
(append '(1 2 3 4 5 6) '(7 8 9))   ===> (1 2 3 4 5 6 7 8 9)
```

Per verificare che un oggetto sia una lista, è disponibile il predicato '**list?**'. Si osservi l'esempio seguente, con il quale si intende ribadire il significato dell'apostrofo per evitare che una lista sia interpretata come funzione:

```
(define a (+ 1 2))
a                               ===> 3

(define b '(+ 1 2))
b                               ===> (+ 1 2)

(list? a)                       ===> #f
(list? b)                       ===> #t
```

Funzioni tipiche sulle liste

Dal momento che con le liste di Scheme non è disponibile un accesso diretto all'elemento ***n***-esimo, se non attraverso la funzione di libreria '**`list-ref`**', è importante imparare a gestire le funzioni elementari già mostrate nella sezione precedente.

- Calcolo della lunghezza di una lista:

```
(define (lunghezza x)
    (if (null? x)
        ; se la lista è vuota, restituisce zero
        0
        ; altrimenti esegue una chiamata ricorsiva
        (+ 1 (lunghezza (cdr x)))
    )
)
```

- Ricerca dell'elemento ***i***-esimo, dove il primo è il numero uno (si veda anche la funzione di libreria '**`list-ref`**', descritta più avanti in questa serie di esempi):

```
(define (i-esimo-elemento i x)
    ; «i» è l'indice, «x» è la lista
    (if (null? x)
        ; la lista è più corta di «i» elementi
        "errore: la lista è troppo corta"
        ; altrimenti procede
        (if (= i 1)
            ; se si tratta del primo elemento, basta la funzione
            ; car per prelevarlo
            (car x)
            ; altrimenti, si utilizza una chiamata ricorsiva
            (i-esimo-elemento (- i 1) (cdr x))
        )
    )
)
```

- Estrae l'ultimo elemento:

```
(define (ultimo x)
    (if (null? x)
        ; la lista è vuota e questo è un errore
        "errore: la lista è vuota"
        ; altrimenti si occupa di estrarre l'ultimo elemento
        (if (null? (cdr x))
            ; se si tratta di una lista contenente un solo elemento,
            ; restituisce il primo e unico di questa
            (car x)
            ; altrimenti utilizza una chiamata ricorsiva
            (ultimo (cdr x))
        )
    )
)
```

- Elimina l'ultimo elemento:

```
(define (elimina-ultimo x)
    (if (null? x)
        ; la lista è vuota e questo è un errore
        "errore: la lista è vuota"
        ; altrimenti si occupa di eliminare l'ultimo elemento
        (if (null? (cdr x))
            ; se si tratta di una lista contenente un solo elemento,
            ; restituisce la lista vuota
            '()
            ; altrimenti utilizza una chiamata ricorsiva per comporre
            ; una lista senza l'ultimo elemento
            (cons (car x) (elimina-ultimo (cdr x)))
        )
    )
)
```

- Restituisce la parte finale della lista, escludendo alcuni elementi iniziali. Si tratta precisamente di una funzione di libreria di Scheme, denominata '**`list-tail`**':

```
(define (list-tail x k)
    (if (zero? k)
        ; se «k» è pari a zero, viene restituita tutta la lista
        x
        ; altrimenti occorre eliminare k-1 elementi iniziali
        ; da (cdr x)
        (list-tail (cdr x) (- k 1))
    )
)
```

- Ricerca del (***k***+1)-esimo elemento di una lista. Si tratta di una funzione di libreria di Scheme, denominata '**`list-ref`**' (in pratica, l'indice ***k*** viene usato in modo da indicare il primo elemento con il numero zero):

```
(define (list-ref x k)
    ; si limita a restituire il primo elemento ottenuto
    ; dalla funzione list-tail
    (car (list-tail x k))
)
```

- Scansione di una lista in modo da restituire un'altra lista, contenente i valori restituiti dalla chiamata di una funzione data per ogni elemento della lista. Si tratta di una semplificazione della funzione di libreria '**`map`**', in questo caso con la possibilità di indicare una sola lista di valori di partenza:

```
(define (map1 f x)
    ; «f» è la funzione da applicare agli elementi della lista «x»
    (if (null? x)
        ; la lista è vuota e restituisce un'altra lista vuota
        '()
        ; altrimenti compone la lista da restituire
        (cons (f (car x)) (map1 f (cdr x)))
    )
)
```

- Descrizione della funzione di libreria '**`append`**':

```
(define (append x y)
    (if (null? x)
        ; se la lista «x» è vuota, restituisce la lista «y»
        y
        ; altrimenti costruisce la lista in modo ricorsivo
        (cons
            (car x)
            (append (cdr x) y)
        )
    )
)
```

- Descrizione della funzione di libreria '**`reverse`**':

```
(define (reverse x)
    (if (null? x)
        ; se la lista «x» è vuota, non c'è nulla da invertire
        '()
        ; altrimenti compone l'inversione con una chiamata ricorsiva
        (append (reverse (cdr x)) (list (car x)))
    )
)
```

Vettori

Scheme gestisce anche i vettori, che sono in pratica gli array dei linguaggi di programmazione normali. Un vettore viene rappresentato in forma costante come una lista preceduta dal simbolo '**`#`**':

`#(` ***elemento_1*** **…** ***elemento_n*** **`)`**

L'indice dei vettori di Scheme parte da zero. Il funzionamento dei vettori di Scheme non richiede spiegazioni particolari. La tabella u129.21 riassume le funzioni utili con questo tipo di dati.

Tabella u129.21. Elenco di alcune funzioni specifiche per la gestione dei vettori.

Funzione	Descrizione
`(vector?` ***oggetto*** `)`	*Vero* se l'oggetto è un vettore.
`(make-vector` ***k*** `)`	Restituisce un vettore di ***k*** elementi indefiniti.
`(make-vector` ***k valore*** `)`	Restituisce un vettore di ***k*** elementi inizializzati al valore specificato.
`(vector` ***elemento…*** `)`	Restituisce un vettore degli elementi indicati.
`(vector-length` ***vettore*** `)`	Restituisce il numero di elementi del vettore.
`(vector-ref` ***vettore k*** `)`	Restituisce l'elemento nella posizione ***k***, partendo da zero.
`(vector-set!` ***vettore k oggetto*** `)`	Assegna all'elemento ***k***-esimo l'oggetto indicato.
`(vector->list` ***vettore*** `)`	Converte il vettore in lista.
`(list->vector` ***lista*** `)`	Converte la lista in vettore.

Strutture di controllo applicate alle liste

Alcune funzioni tipiche di Scheme servono ad applicare una funzione a un gruppo di valori contenuto in una lista.

Tabella u129.22. Elenco di alcune funzioni specifiche per la scansione degli elementi di una lista, allo scopo di applicarvi una funzione.

Funzione	Descrizione
`(apply` ***funzione lista*** `)`	Esegue la funzione utilizzando gli elementi della lista come argomenti.
`(map` ***funzione lista…*** `)`	Esegue la funzione iterativamente per gli elementi delle liste.
`(for-each` ***funzione lista…*** `)`	Esegue la funzione iterativamente per gli elementi delle liste.

Funzione «apply»

```
(apply funzione lista)
```

La funzione '**`apply`**' esegue una funzione a cui affida gli elementi di una lista come altrettanti argomenti. Si osservi il modello seguente:

```
(apply funzione '(elem_1 elem_2… elem_n))
```

Questo equivale in pratica a:

```
(funzione elem_1 elem_2… elem_n)
```

Per esempio:

```
(apply + '(1 2))                    ===> 3
```

Funzione «map»

```
(map funzione lista…)
```

La funzione '**`map`**' scandisce una o più liste, tutte con la stessa quantità di elementi, in modo tale che, a ogni ciclo, viene passato alla funzione l'insieme ordinato dell'***i***-esimo elemento di ognuna di queste liste. La funzione restituisce una lista contenente i valori restituiti dalla funzione a ogni ciclo.

Anche se viene rispettato l'ordine delle varie liste, '**`dat`**' non garantisce che la scansione avvenga dal primo elemento all'ultimo.

L'esempio seguente esegue la somma di una serie di coppie di valori, restituendo la lista dei risultati:

```
(map + '(1 2 3) '(4 5 6))                ===> (5 7 9)
```

Funzione «for-each»

```
(for-each funzione lista…)
```

La funzione '**`for-each`**' è molto simile a '**`map`**', nel senso che avvia una funzione ripetutamente, quanti sono gli elementi delle liste successive, garantendo di eseguire l'operazione in ordine, secondo la sequenza degli elementi nelle liste. Tuttavia, non restituisce nulla.

Riferimenti

- A. Aaby, *Scheme Tutorial*, 1996
  http://cs.wwc.edu/~cs_dept/KU/PR/Scheme.html
- *R*^{5}*RS -- Revised-5 Report on the Algorithmic Language Scheme*, 1998
  http://www.swiss.ai.mit.edu/~jaffer/r5rs_toc.html
  http://www.swiss.ai.mit.edu/ftpdir/scheme-reports/r5rs.ps.gz

[1] A questo punto si intende ormai chiarito il significato dell'apostrofo posto di fronte a una lista, quando questa non deve essere valutata, prima di essere fornita come argomento di una funzione.

Scheme: I/O

Apertura e chiusura 1075
Ingresso dei dati 1075
Uscita dei dati 1076

Scheme ha una gestione particolare dei file. Per prima cosa, i flussi di file, che negli altri linguaggi sono dei *file handle* o semplicemente *stream*, in Scheme prendono il nome di *port*: ***porte***. Scheme distingue quindi tra porte in ingresso, in grado di «consegnare» dei caratteri, e porte in uscita, in grado di «accettare» caratteri.

Apertura e chiusura

Scheme distingue tra flussi di file in ingresso e in uscita, per cui le funzioni per aprire i file e trasformarli in porte, sono due, uno per l'apertura in lettura (ingresso) e l'altra per l'apertura in scrittura (uscita). La tabella u130.1 riassume le funzioni utili per aprire, controllare e chiudere i file. Gli esempi successivi, dovrebbero aiutare a comprenderne l'utilizzo.

Tabella u130.1. Elenco di alcune funzioni per l'apertura e la chiusura dei file, oltre che per il controllo dei flussi di file predefiniti.

Funzione	Descrizione
`(open-input-file` ***str_nome_file*** `)`	Apre il file nominato e restituisce la porta in ingresso.
`(open-output-file` ***str_nome_file*** `)`	Apre il file nominato e restituisce la porta in uscita.
`(port?` ***oggetto*** `)`	*Vero* se si tratta di una porta.
`(input-port?` ***oggetto*** `)`	*Vero* se si tratta di una porta in ingresso.
`(output-port?` ***oggetto*** `)`	*Vero* se si tratta di una porta in uscita.
`(close-input-port` ***porta*** `)`	Chiude la porta in ingresso.
`(close-output-port` ***porta*** `)`	Chiude la porta in uscita.

```
(define porta-i (open-input-file "mio_file"))

(port? porta-i)                          ===> #t
(output-port? porta-i)                   ===> #f
(input-port? porta-i)                    ===> #t

(close-input-port porta-i)
```

In condizioni normali, sono sempre disponibili una porta in ingresso e una in uscita, in modo predefinito. Si tratta generalmente di standard input e standard output. Questi flussi di file predefiniti potrebbero essere diretti verso altri file. Tuttavia questo non viene mostrato; eventualmente si può approfondire il problema leggendo *R*^{5}*RS*.

Ingresso dei dati

L'ingresso dei dati, ovvero la lettura, avviene attraverso due funzioni fondamentali: '**`read-char`**' e '**`read`**'. La prima legge un carattere alla volta, la seconda interpreta ciò che legge in forma di dati Scheme. In pratica, '**`read`**' legge ogni volta ciò che riesce a interpretare come un oggetto per Scheme.

Tabella u130.3. Elenco di alcune funzioni per la gestione dei dati in ingresso.

Funzione	Descrizione
`(read-char)`	Legge e restituisce il carattere successivo dalla porta predefinita.
`(read-char` ***porta*** `)`	Legge e restituisce il carattere successivo dalla porta indicata.
`(peek-char)`	Restituisce una copia del carattere successivo dalla porta predefinita.

Funzione	Descrizione
`(peek-char` ***porta*** `)`	Restituisce una copia del carattere successivo dalla porta indicata.
`(read)`	Legge un oggetto dalla porta predefinita.
`(read` ***porta*** `)`	Legge un oggetto dalla porta indicata.
`(eof-object` ***porta*** `)`	*Vero* la lettura dalla porta ha raggiunto la fine.

L'esempio seguente mostra in che modo potrebbe essere utilizzata la funzione '**`read-char`**'. Si inizia aprendo il file '`/etc/passwd`', dal quale vengono letti i primi caratteri. Si suppone che il primo record a essere letto sia quello di definizione dell'utente '**`root`**':

```
; apre il file e gli associa la porta «utenti»
(define utenti (open-input-file "/etc/passwd"))

; legge un carattere alla volta
(read-char utenti)                          ===> #\r
(read-char utenti)                          ===> #\o
(read-char utenti)                          ===> #\o
(read-char utenti)                          ===> #\t
(read-char utenti)                          ===> #\:
;…

; chiude il file
(close-input-file utenti)
```

Nell'esempio seguente si vuole mostrare l'uso della funzione '**`read`**'. Prima si suppone di avere preparato il file seguente:

```
; prova_lettura.scm

; somma
(+ 1 2)

; moltiplicazione
(* 2 5)

; stringa
"ciao"

; valore numerico
123

; fine
```

Supponendo che il file si chiami '`prova_lettura.scm`', si osservi la sequenza di istruzioni Scheme seguente, assieme a ciò che si ottiene dalla lettura del file:

```
; apre il file e gli associa la porta «prova»
(define prova (open-input-file "prova_lettura.scm"))

; legge il primo oggetto
(read utenti)                               ===> (+ 1 2)

; legge il secondo oggetto
(read utenti)                               ===> (* 2 5)

; legge il terzo oggetto
(read utenti)                               ===> "ciao"

; legge il quarto oggetto
(read utenti)                               ===> 123

; chiude il file
(close-input-file prova)
```

Si intende l'importanza della funzione '**`read`**' per facilitare l'inserimento di dati nei programmi in modo interattivo.

Uscita dei dati

«

L'emissione dei dati, ovvero la scrittura, avviene in maniera simile alla lettura, con la stessa distinzione tra le funzioni '**`write-char`**' e '**`write`**'. Anche in questo caso, la prima scrive un carattere alla volta, mentre la seconda emette la rappresentazione di un oggetto alla volta. Tuttavia, si aggiunte un'altra funzione fondamentale: '**`output`**'. Questa funzione viene usata preferibilmente per mostrare dei messaggi senza codici di escape, soprattutto per non lasciare le virgolette di delimitazione delle stringhe.

Tabella u130.7. Elenco di alcune funzioni per la gestione dei dati in ingresso.

Funzione	Descrizione
`(write-char` ***carattere*** `)`	Scrive il carattere indicato attraverso la porta predefinita.
`(write-char` ***carattere porta*** `)`	Scrive il carattere indicato attraverso la porta indicata.
`(write` ***oggetto*** `)`	Scrive la rappresentazione dell'oggetto attraverso la porta predefinita.
`(write` ***oggetto porta*** `)`	Scrive la rappresentazione dell'oggetto attraverso la porta indicata.
`(display` ***oggetto*** `)`	Mostra l'oggetto attraverso la porta predefinita.
`(display` ***oggetto porta*** `)`	Mostra l'oggetto attraverso la porta indicata.
`(newline)`	Emette un codice di interruzione di riga attraverso la porta predefinita.
`(newline` ***porta*** `)`	Emette un codice di interruzione di riga attraverso la porta indicata.

L'esempio seguente dovrebbe chiarire la differenza tra la funzione '**`write`**' e '**`display`**'. Gli oggetti vengono emessi attraverso lo standard output, ovvero la porta predefinita:

```
(write (+ 1 2))                         ; visualizza «3»
(write "ciao")                          ; visualizza «"ciao"»
(write "ciao, come \"stai\"?")          ; visualizza «"ciao, come \"stai\""»
(write #\A)                             ; visualizza «#\A»
(display (+ 1 2))                       ; visualizza «3»
(display "ciao")                        ; visualizza «ciao»
(display "ciao, come \"stai\"?")        ; visualizza «ciao, come "stai"»
(display #\A)                           ; visualizza «A»
```

È già stato descritto l'uso di '**`newline`**', che è indispensabile per ottenere l'avanzamento alla riga successiva. In linea di principio, non è possibile inserire un carattere di controllo nella stringa emessa da '**`write`**' o da '**`display`**'.

Problemi elementari di programmazione 1079
Somma tra due numeri positivi 1079
Moltiplicazione di due numeri positivi attraverso la somma 1080
Divisione intera tra due numeri positivi 1081
Elevamento a potenza 1081
Radice quadrata 1082
Fattoriale 1083
Massimo comune divisore 1083
Numero primo 1084
Scansione di array 1084
Ricerca sequenziale 1084
Ricerca binaria 1085
Algoritmi tradizionali 1086
Bubblesort 1086
Torre di Hanoi 1087
Quicksort 1087
Permutazioni 1089

Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Lo scopo di questi esempi è solo didattico, utilizzando forme non ottimizzate per la velocità di esecuzione.

Problemi elementari di programmazione 1079
Somma tra due numeri positivi 1079
Moltiplicazione di due numeri positivi attraverso la somma 1080
Divisione intera tra due numeri positivi 1081
Elevamento a potenza 1081
Radice quadrata 1082
Fattoriale 1083
Massimo comune divisore 1083
Numero primo 1084
Scansione di array 1084
Ricerca sequenziale 1084
Ricerca binaria 1085
Algoritmi tradizionali 1086
Bubblesort 1086
Torre di Hanoi 1087
Quicksort 1087
Permutazioni 1089

Problemi elementari di programmazione

In questa sezione vengono mostrati alcuni algoritmi elementari portati in Scheme.

Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
; ====================================================================
; somma1.scm
; Somma esclusivamente valori positivi.
; ====================================================================

; ====================================================================
; (somma <x> <y>)
; --------------------------------------------------------------------
(define (somma x y)
    (define z x)
```

```
    (define i 1)

    (do ()
        ((> i y))

        (set! z (+ z 1))
        (set! i (+ i 1))
    )

    z
)

; ================================================================
; Inizio del programma.
; ----------------------------------------------------------------
(define x 0)
(define y 0)
(define z 0)

(display "Inserisci il primo numero intero positivo: ")
(set! x (read))
(newline)
(display "Inserisci il secondo numero intero positivo: ")
(set! y (read))
(newline)
(set! z (somma x y))
(display x) (display " + ") (display y) (display " = ") (display z)
(newline)

; ================================================================
```

In alternativa, si può modificare la funzione '**somma**', in modo che il ciclo '**do**' gestisca la dichiarazione e l'incremento delle variabili che usa. Tuttavia, in questo caso, la variabile '**z**' deve essere «copiata» in modo da poter trasmettere il risultato all'esterno del ciclo '**do**':

```
(define (somma x y)
    (define risultato 0)

    (do ((z x (+ z 1)) (i 1 (+ i 1)))
        ((> i y))
        (set! risultato z)
    )

    risultato
)
```

Volendo gestire la cosa in modo un po' più elegante, occorre togliere la variabile '**z**' dalla gestione del ciclo '**do**':

```
(define (somma x y)
    (define z x)

    (do ((i 1 (+ i 1)))
        ((> i y))
        (set! z (+ z 1))
    )

    z
)
```

Moltiplicazione di due numeri positivi attraverso la somma

«

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
; ================================================================
; moltiplica1.scm
; Moltiplica esclusivamente valori positivi.
; ================================================================

; ================================================================
; (moltiplica <x> <y>)
; ----------------------------------------------------------------
(define (moltiplica x y)
    (define z 0)
    (define i 1)

    (do ()
        ((> i y))

        (set! z (+ z x))
        (set! i (+ i 1))
    )

    z
)

; ================================================================
; Inizio del programma.
; ----------------------------------------------------------------
(define x 0)
(define y 0)
(define z 0)

(display "Inserisci il primo numero intero positivo: ")
(set! x (read))
(newline)
(display "Inserisci il secondo numero intero positivo: ")
(set! y (read))
(newline)
```



```
(set! z (moltiplica x y))
(display x) (display " * ") (display y) (display " = ") (display z)
(newline)

; ================================================================
```

In alternativa, si può modificare la funzione '**moltiplica**', in modo che il ciclo '**do**' gestisca la dichiarazione e l'incremento dell'indice '**i**':

```
(define (moltiplica x y)
    (define z 0)

    (do ((i 1 (+ i 1)))
        ((> i y))

        (set! z (+ z x))
    )

    z
)
```

Divisione intera tra due numeri positivi

«

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
; ================================================================
; dividi1.scm
; Divide esclusivamente valori positivi.
; ================================================================

; ================================================================
; (dividi <x> <y>)
; ----------------------------------------------------------------
(define (dividi x y)
    (define z 0)
    (define i x)

    (do ()
        ((< i y))

        (set! i (- i y))
        (set! z (+ z 1))
    )

    z
)

; ================================================================
; Inizio del programma.
; ----------------------------------------------------------------
(define x 0)
(define y 0)
(define z 0)

(display "Inserisci il primo numero intero positivo: ")
(set! x (read))
(newline)
(display "Inserisci il secondo numero intero positivo: ")
(set! y (read))
(newline)
(set! z (dividi x y))
(display x) (display " / ") (display y) (display " = ") (display z)
(newline)

; ================================================================
```

In alternativa, si può modificare la funzione '**dividi**', in modo che il ciclo '**do**' gestisca la dichiarazione e il decremento della variabile '**i**'. Per la precisione, la variabile '**z**' non può essere dichiarata nello stesso modo, perché serve anche al di fuori del ciclo:

```
(define (dividi x y)
    (define z 0)

    (do ((i x (- i y)))
        ((< i y))

        (set! z (+ z 1))
    )

    z
)
```

Elevamento a potenza

«

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
; ================================================================
; potenza1.scm
; Eleva a potenza.
; ================================================================

; ================================================================
; (potenza <x> <y>)
; ----------------------------------------------------------------
```

```
(define (potenza x y)
    (define z 1)
    (define i 1)

    (do ()
        ((> i y))

        (set! z (* z x))
        (set! i (+ i 1))
    )

    z
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------
(define x 0)
(define y 0)
(define z 0)

(display "Inserisci il primo numero intero positivo: ")
(set! x (read))
(newline)
(display "Inserisci il secondo numero intero positivo: ")
(set! y (read))
(newline)
(set! z (potenza x y))
(display x) (display " ** ") (display y) (display " = ") (display z)
(newline)

; ====================================================================
```

In alternativa, si può modificare la funzione '**`potenza`**', in modo che il ciclo '**`do`**' gestisca la dichiarazione e l'incremento della variabile '**`i`**':

```
(define (potenza x y)
    (define z 1)

    (do ((i 1 (+ i 1)))
        ((> i y))

        (set! z (* z x))
    )

    z
)
```

È possibile usare anche un algoritmo ricorsivo:

```
(define (potenza x y)
    (if (= x 0)
        0
        (if (= y 0)
            1
            (* x (potenza x (- y 1)))
        )
    )
)
```

Radice quadrata

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
; ====================================================================
; radice1.scm
; Radice quadrata.
; ====================================================================

; ====================================================================
; (radice <x>)
; --------------------------------------------------------------------
(define (radice x)
    (define z -1)
    (define t 0)
    (define uscita #f)

    (do ()
        (uscita)

        (set! z (+ z 1))
        (set! t (* z z))
        (if (> t x)
            ; È stato superato il valore massimo
            (begin
                (set! z (- z 1))
                (set! uscita #t)
            )
        )
    )

    z
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------
(define x 0)
(define z 0)

(display "Inserisci il numero intero positivo: ")
```



```
(set! x (read))
(newline)
(set! z (radice x))
(display "La radice quadrata di ") (display x) (display " è ") (display z)
(newline)

; ====================================================================
```

Fattoriale

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
; ====================================================================
; fattoriale1.scm
; Fattoriale.
; ====================================================================

; ====================================================================
; (fattoriale <x>)
; --------------------------------------------------------------------
(define (fattoriale x)
    (define i (- x 1))

    (do ()
        ((<= i 0))

        (set! x (* x i))
        (set! i (- i 1))
    )

    x
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------
(define x 0)
(define z 0)

(display "Inserisci il numero intero positivo: ")
(set! x (read))
(newline)
(set! z (fattoriale x))
(display x) (display "! = ") (display z)
(newline)

; ====================================================================
```

In alternativa, l'algoritmo si può tradurre in modo ricorsivo:

```
(define (fattoriale x)
    (if (> x 1)
        (* x (fattoriale (- x 1)))
        1
    )
)
```

Massimo comune divisore

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione 62.3.7.

```
; ====================================================================
; mcd1.scm
; Massimo Comune Divisore.
; ====================================================================

; ====================================================================
; (moltiplica <x> <y>)
; --------------------------------------------------------------------
(define (mcd x y)
    (do ()
        ((= x y))

        (if (> x y)
            (set! x (- x y))
            (set! y (- y x))
        )
    )

    x
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------
(define x 0)
(define y 0)
(define z 0)

(display "Inserisci il primo numero intero positivo: ")
(set! x (read))
(newline)
(display "Inserisci il secondo numero intero positivo: ")
(set! y (read))
(newline)
(set! z (mcd x y))
(display "MCD di ") (display x) (display " e ") (display y)
(display " è ") (display z)
(newline)
```

```
; =====================================================================
```

Numero primo

Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione 62.3.8.

```
; =====================================================================
; primo1.scm
; Numero primo.
; =====================================================================

; =====================================================================
; (primo <x>)
; ---------------------------------------------------------------------
(define (primo x)
    (define np #t)
    (define i 2)
    (define j 0)

    (do ()
        ((or (>= i x) (not np)))

        (set! j (truncate (/ x i)))
        (set! j (- x (* j i)))
        (if (= j 0)
            (set! np #f)
            (set! i (+ i 1))
        )
    )

    np
)

; =====================================================================
; Inizio del programma.
; ---------------------------------------------------------------------
(define x 0)

(display "Inserisci un numero intero positivo: ")
(set! x (read))
(newline)
(if (primo x)
    (display "È un numero primo")
    (display "Non è un numero primo")
)
(newline)

; =====================================================================
```

Scansione di array

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
; =====================================================================
; ricerca_sequenziale1.scm
; Ricerca Sequenziale.
; =====================================================================

; =====================================================================
; (ricerca <vettore> <x> <ele-inf> <ele-sup>)
; ---------------------------------------------------------------------
(define (ricerca vettore x a z)
    (define risultato -1)

    (do ((i a (+ i 1)))
        ((> i z))

        (if (= x (vector-ref vettore i))
            (set! risultato i)
        )
    )
    risultato
)

; =====================================================================
; Inizio del programma.
; ---------------------------------------------------------------------

(define DIM 100)
(define vettore (make-vector DIM))
(define x 0)
(define i 0)
(define z 0)

(display "Inserire la quantità di elementi; ")
(display DIM)
(display " al massimo: ")
(set! z (read))
(newline)

(if (> z DIM)
    (set! z DIM)
)

(display "Inserire i valori del vettore.")
```



```
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display "elemento ")
    (display i)
    (display " ")
    (vector-set! vettore i (read))
    (newline)
)

(display "Inserire il valore da cercare: ")
(set! x (read))
(newline)

(set! i (ricerca vettore x 0 (- z 1)))

(display "Il valore cercato si trova nell'elemento ")
(display i)
(newline)

; =====================================================================
```

Esiste anche una soluzione ricorsiva che viene mostrata di seguito:

```
(define (ricerca vettore x a z)
    (if (> a z)
        ; La corrispondenza non è stata trovata.
        1
        (if (= x (vector-ref vettore a))
            a
            (ricerca vettore x (+ a 1) z)
        )
    )
)
```

Ricerca binaria

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione 62.4.2.

```
; =====================================================================
; ricerca_binaria1.scm
; Ricerca Binaria.
; =====================================================================

; =====================================================================
; (ricerca <vettore> <x> <ele-inf> <ele-sup>)
; ---------------------------------------------------------------------
(define (ricerca vettore x a z)
    (define m (truncate (/ (+ a z) 2)))

    (if (or (< m a) (> m z))
        ; Non restano elementi da controllare: l'elemento cercato
        ; non c'è.
        -1

        (if (< x (vector-ref vettore m))
            ; Si ripete la ricerca nella parte inferiore.
            (ricerca vettore x a (- m 1))

            (if (> x (vector-ref vettore m))
                ; Si ripete la ricerca nella parte superiore.
                (ricerca vettore x (+ m 1) z)

                ; Se x è uguale a vettore[m], l'obiettivo è
                ; stato trovato.
                m
            )
        )
    )
)

; =====================================================================
; Inizio del programma.
; ---------------------------------------------------------------------

(define DIM 100)
(define vettore (make-vector DIM))
(define x 0)
(define i 0)
(define z 0)

(display "Inserire la quantità di elementi; ")
(display DIM)
(display " al massimo: ")
(set! z (read))
(newline)

(if (> z DIM)
    (set! z DIM)
)

(display "Inserire i valori del vettore (in modo ordinato).")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display "elemento ")
    (display i)
    (display " ")
    (vector-set! vettore i (read))
    (newline)
```

```
)

(display "Inserire il valore da cercare: ")
(set! x (read))
(newline)

(set! i (ricerca vettore x 0 (- z 1)))

(display "Il valore cercato si trova nell'elemento ")
(display i)
(newline)

; ================================================================
```

Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in Scheme.

Bubblesort

Il problema del Bubblesort è stato descritto nella sezione 62.5.1. Viene mostrato prima una soluzione iterativa e successivamente la funzione '**bsort**' in versione ricorsiva.

```
; ================================================================
; bsort1.scm
; Bubblesort.
; ================================================================

; ================================================================
; (ordina <vettore> <ele-inf> <ele-sup>)
; ----------------------------------------------------------------
(define (ordina vettore a z)
    (define scambio 0)

    (do ((j a (+ j 1)))
        ((>= j z))

        (do ((k (+ j 1) (+ k 1)))
            ((> k z))

            (if (< (vector-ref vettore k) (vector-ref vettore j))
                ; Scambia i valori.
                (begin
                    (set! scambio (vector-ref vettore k))
                    (vector-set! vettore k (vector-ref vettore j))
                    (vector-set! vettore j scambio)
                )
            )
        )
    )

    vettore
)

; ================================================================
; Inizio del programma.
; ----------------------------------------------------------------

(define DIM 100)
(define vettore (make-vector DIM))
(define x 0)
(define i 0)
(define z 0)

(display "Inserire la quantità di elementi; ")
(display DIM)
(display " al massimo: ")
(set! z (read))
(newline)

(if (> z DIM)
    (set! z DIM)
)

(display "Inserire i valori del vettore.")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display "elemento ")
    (display i)
    (display " ")
    (vector-set! vettore i (read))
    (newline)
)

(set! vettore (ordina vettore 0 (- z 1)))

(display "Il vettore ordinato è il seguente: ")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display (vector-ref vettore i))
    (display " ")
)
(newline)

; ================================================================
```


Segue la funzione '**ordina**' in versione ricorsiva:

```
(define (ordina vettore a z)
    (define scambio 0)

    (if (< a z)
        (begin
            ; Scansione interna dell'array per collocare nella
            ; posizione a l'elemento giusto.
            (do ((k (+ a 1) (+ k 1)))
                ((> k z))

                (if (< (vector-ref vettore k) (vector-ref vettore a))
                    ; Scambia i valori.
                    (begin
                        (set! scambio (vector-ref vettore k))
                        (vector-set! vettore k (vector-ref vettore a))
                        (vector-set! vettore a scambio)
                    )
                )
            )

            (set! vettore (ordina vettore (+ a 1) z))
        )
    )
    vettore
)
```

Torre di Hanoi

Il problema della torre di Hanoi è descritto nella sezione 62.5.3.

```
; ================================================================
; hanoi1.scm
; Torre di Hanoi.
; ================================================================

; ================================================================
; (hanoi <n-anelli> <piolo-iniziale> <piolo-finale>)
; ----------------------------------------------------------------
(define (hanoi n p1 p2)
    (if (> n 0)
        (begin
            (hanoi (- n 1) p1 (- 6 (+ p1 p2)))
            (begin
                (display "Muovi l'anello ")
                (display n)
                (display " dal piolo ")
                (display p1)
                (display " ")
                (display p2)
                (newline)
            )
            (hanoi (- n 1) (- 6 (+ p1 p2)) p2)
        )
    )
)


; ================================================================
; Inizio del programma.
; ----------------------------------------------------------------
(define n 0)
(define p1 0)
(define p2 0)

(display "Inserisci il numero di pioli: ")
(set! n (read))
(newline)
(display "Inserisci il numero del piolo iniziale (da 1 a 3): ")
(set! p1 (read))
(newline)
(display "Inserisci il numero del piolo finale (da 1 a 3): ")
(set! p2 (read))
(newline)
(hanoi n p1 p2)

; ================================================================
```

Quicksort

L'algoritmo del Quicksort è stato descritto nella sezione 62.5.4.

```
; ================================================================
; qsort1.scm
; Quicksort.
; ================================================================

; ----------------------------------------------------------------
; Dichiara il vettore a cui successivamente fanno riferimento tutte le
; funzioni.
; Il vettore non viene passato alle funzioni tra gli argomenti, per
; semplificare le funzioni, soprattutto nel caso di «part», che
; deve restituire anche un altro valore.
; ----------------------------------------------------------------
(define DIM 100)
(define vettore (make-vector DIM))

; ================================================================
; (inverti-elementi <indice-1> <indice-2>)
; ----------------------------------------------------------------
(define (inverti-elementi a z)
```

```
    (define scambio 0)
    (set! scambio (vector-ref vettore a))
    (vector-set! vettore a (vector-ref vettore z))
    (vector-set! vettore z scambio)
)

; ====================================================================
; (part <ele-inf> <ele-sup>)
; --------------------------------------------------------------------
(define (part a z)
    ; Si assume che «a» sia inferiore a «z».
    (define i (+ a 1))
    (define cf z)
    ; Vengono preparate delle variabili per controllare l'uscita dai cicli.
    (define uscita1 #f)
    (define uscita2 #f)
    (define uscita3 #f)

    ; Inizia il ciclo di scansione dell'array.
    (set! uscita1 #f)
    (do ()
        (uscita1)
        (set! uscita2 #f)
        (do ()
            (uscita2)

            ; Sposta «i» a destra.
            (if (or
                    (> (vector-ref vettore i) (vector-ref vettore a))
                    (>= i cf)
                )
                ; Interrompe il ciclo interno.
                (set! uscita2 #t)
                ; Altrimenti incrementa l'indice
                (set! i (+ i 1))
            )
        )
        (set! uscita3 #f)
        (do ()
            (uscita3)

            ; Sposta «cf» a sinistra.
            (if (<= (vector-ref vettore cf) (vector-ref vettore a))
                ; Interrompe il ciclo interno.
                (set! uscita3 #t)
                ; Altrimenti decrementa l'indice
                (set! cf (- cf 1))
            )
        )

        (if (<= cf i)
            ; È avvenuto l'incontro tra «i» e «cf».
            (set! uscita1 #t)
            ; Altrimenti vengono scambiati i valori.
            (begin
                (inverti-elementi i cf)
                (set! i (+ i 1))
                (set! cf (- cf 1))
            )
        )
    )

    ; A questo punto vettore[a..z] è stato ripartito e «cf» è la
    ; collocazione di vettore[a].
    (inverti-elementi a cf)

    ; A questo punto, vettore[cf] è un elemento (un valore) nella
    ; posizione giusta, e «cf» è ciò che viene restituito.
    cf
)

; ====================================================================
; (ordina <ele-inf> <ele-sup>)
; --------------------------------------------------------------------
(define (ordina a z)
    ; Viene preparata la variabile «cf».
    (define cf 0)

    (if (> z a)
        (begin
            (set! cf (part a z))
            (ordina a (- cf 1))
            (ordina (+ cf 1) z)
        )
    )
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------

(define x 0)
(define i 0)
(define z 0)

(display "Inserire la quantità di elementi; ")
(display DIM)
(display " al massimo: ")
(set! z (read))
(newline)

(if (> z DIM)
    (set! z DIM)
)

(display "Inserire i valori del vettore.")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display "elemento ")
    (display i)
    (display " ")
    (vector-set! vettore i (read))
    (newline)
)

; Il vettore non viene trasferito come argomento della funzione,
; ma risulta accessibile esternamente.
(ordina 0 (- z 1))

(display "Il vettore ordinato è il seguente: ")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i z))

    (display (vector-ref vettore i))
    (display " ")
)
(newline)

; ====================================================================
```


Permutazioni

L'algoritmo ricorsivo delle permutazioni è descritto nella sezione 62.5.5.

```
; ====================================================================
; permuta1.scm
; Permutazioni.
; ====================================================================

; --------------------------------------------------------------------
; Dichiara il vettore a cui successivamente fanno riferimento tutte le
; funzioni.
; --------------------------------------------------------------------
(define DIM 100)
(define vettore (make-vector DIM))

; --------------------------------------------------------------------
; Sempre per motivi pratici, rende disponibile la dimensione utilizzata
; effettivamente.
; --------------------------------------------------------------------
(define n-elementi 0)

; ====================================================================
; (inverti-elementi <indice-1> <indice-2>)
; --------------------------------------------------------------------
(define (inverti-elementi a z)
    (define scambio 0)
    (set! scambio (vector-ref vettore a))
    (vector-set! vettore a (vector-ref vettore z))
    (vector-set! vettore z scambio)
)

; ====================================================================
; (visualizza)
; --------------------------------------------------------------------
(define (visualizza)
    (do ((i 0 (+ i 1)))
        ((>= i n-elementi))

        (display (vector-ref vettore i))
        (display " ")
    )
    (newline)
)

; ====================================================================
; (permuta <inizio> <fine>)
; --------------------------------------------------------------------
(define (permuta a z)
    (define k 0)

    ; Se il segmento di array contiene almeno due elementi, si
    ; procede.
    (if (>= (- z a) 1)
        ; Inizia un ciclo di scambi tra l'ultimo elemento e uno
        ; degli altri contenuti nel segmento di array.
        (do ((k z (- k 1)))
            ((< k a))

            ; Scambia i valori.
            (inverti-elementi k z)

            ; Esegue una chiamata ricorsiva per permutare un
            ; segmento più piccolo dell'array.
            (permuta a (- z 1))

            ; Scambia i valori.
            (inverti-elementi k z)
        )

        ; Altrimenti, visualizza l'array e utilizza una variabile
```

```
        ; dichiarata globalmente.
        (visualizza)
    )
)

; ====================================================================
; Inizio del programma.
; --------------------------------------------------------------------
(display "Inserire la quantità di elementi; ")
(display DIM)
(display " al massimo: ")
(set! n-elementi (read))
(newline)

(if (> n-elementi DIM)
    (set! n-elementi DIM)
)

(display "Inserire i valori del vettore.")
(newline)
(do ((i 0 (+ i 1)))
    ((>= i n-elementi))

    (display "elemento ")
    (display i)
    (display " ")
    (vector-set! vettore i (read))
    (newline)
)

; Il vettore non viene trasferito come argomento della funzione,
; ma risulta accessibile esternamente.
(permuta 0 (- n-elementi 1))

; ====================================================================
```


Parte xxx

BC: linguaggio aritmetico a precisione arbitraria

«

BC: esempi di programmazione 1093
Problemi elementari di programmazione 1094
Scansione di array 1096
Algoritmi tradizionali 1097

Problemi elementari di programmazione 1094
Somma tra due numeri positivi 1094
Moltiplicazione di due numeri positivi attraverso la somma 1094
Divisione intera tra due numeri positivi 1094
Elevamento a potenza 1095
Radice quadrata 1095
Fattoriale 1095
Massimo comune divisore 1096
Numero primo 1096
Scansione di array 1096
Ricerca sequenziale 1097
Ricerca binaria 1097
Algoritmi tradizionali 1097
Bubblesort 1098
Torre di Hanoi 1098
Quicksort 1098
Permutazioni 1099

BC, ovvero *Basic calculator*, è un interprete di un linguaggio aritmetico, descritto nella sezione 22.14. Questo capitolo raccoglie solo alcuni esempi di programmazione, in parte già descritti in altri capitoli. Per eseguire questi esempi basta usare il comando seguente, dove '`prova.b`' rappresenta il nome del file da eseguire:

```
$ bc prova.b [Invio]
```

Si vuole evitare l'uso di estensioni al linguaggio BC, per cui i programmi non vengono mostrati come script; inoltre manca la possibilità di controllare l'interazione con l'utilizzatore, quindi le funzioni devono essere richiamate manualmente e al termine si deve usare il comando '**`quit`**', oppure si conclude il flusso dello standard input con la combinazione [*Ctrl d*].

Negli esempi non si fa uso delle librerie standard, pertanto i nomi relativi possono essere riutilizzati.

Le espressioni vengono scritte in modo da evitare la visualizzazione non desiderata. Per esempio, invece di '**`i++`**', si preferisce usare la forma '**`i=(i+1)`**', quando possibile.

Bisogna ricordare che se non si assegna il risultato generato da una funzione, questo viene visualizzato. La variabile '**`t`**' è stata usata negli esempi per raccogliere questo risultato quando non desiderato.

Problemi elementari di programmazione 1094
Somma tra due numeri positivi 1094
Moltiplicazione di due numeri positivi attraverso la somma 1094
Divisione intera tra due numeri positivi 1094
Elevamento a potenza 1095
Radice quadrata 1095
Fattoriale 1095
Massimo comune divisore 1096
Numero primo 1096
Scansione di array 1096
Ricerca sequenziale 1097
Ricerca binaria 1097
Algoritmi tradizionali 1097

Bubblesort 1098
Torre di Hanoi 1098
Quicksort 1098
Permutazioni 1099

Problemi elementari di programmazione

In questa sezione vengono mostrati alcuni algoritmi elementari portati in BC. Per la spiegazione degli algoritmi, se non sono già conosciuti, occorre leggere quanto riportato nel capitolo ??capitolo programmazione pseudo??.

Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
/*
    somma.b
    Somma esclusivamente valori positivi.
*/

define s (x, y) {
    auto z, i
    z=x
    for (i=1; i<=y; i++) {
        z=(z+1)
    }
    return (z)
}

"Per calcolare la somma, si utilizzi la funzione s (x, y): "
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
define s (x, y) {
    auto z, i
    z=x
    i=1
    while (i<=y) {
        z=(z+1)
        i=(i+1)
    }
    return (z)
}
```

Moltiplicazione di due numeri positivi attraverso la somma

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
/*
    moltiplica.b
*/

define m (x, y) {
    auto z, i
    z=0
    for (i=1; i<=y; i++) {
        z=(z+x)
    }
    return (z)
}

"Per calcolare la moltiplicazione, si utilizzi la funzione m (x, y): "
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
define m (x, y) {
    auto z, i
    z=0
    i=1
    while (i<=y) {
        z=(z+x)
        i=(i+1)
    }
    return (z)
}
```

Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
/*
    dividi.b
    Divide esclusivamente valori positivi.
*/

define d (x, y) {
    auto z, i
    z=0
    i=x
    while (i>=y) {
        i=(i-y)
        z=(z+1)
    }
    return (z)
}

"Per calcolare la divisione intera, si utilizzi la funzione d (x, y): "
```

Elevamento a potenza

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
/*
    exp.b
*/

define x (x, y) {
    auto z, i
    z=1
    for (i=1; i<=y; i++) {
        z=(z*x)
    }
    return (z)
}

"Per calcolare l'elevamento a potenza, si utilizzi la funzione x (x, y): "
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**':

```
define x (x, y) {
    auto z, i
    z=1
    i=1
    while (i<=y) {
        z=(z*x)
        i=(i+1)
    }
    return (z)
}
```

È possibile usare anche un algoritmo ricorsivo:

```
define x (x, y) {
    if (x==0) {
        return (0)
    }
    if (y==0) {
        return (1)
    }
    return (x * x (x, y-1))
}
```

Radice quadrata

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
/*
    radice.b
*/

define r (x) {
    auto z, y
    z=0
    y=0
    while (1) {
        y=(z*z)
        if (y>x) {
            /* È stato superato il valore massimo. */
            z=(z-1)
            return (z)
        }
        z=(z+1)
    }
}

"Per calcolare la radice quadrata, si utilizzi la funzione r (x): "
```

Fattoriale

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
/*
    fatt.b
*/

define f (x) {
    auto i
    i=(x-1)
    while (i>0) {
        x=(x*i)
        i=(i-1)
    }
    return (x)
}

"Per calcolare il fattoriale, si utilizzi la funzione f (x): "
```

In alternativa, l'algoritmo si può tradurre in modo ricorsivo:

```
define f (x) {
    if (x>1) {
        return (x * f (x-1))
    }
    return (1)
}
```

Massimo comune divisore

Il problema del massimo comune divisore, tra due numeri positivi, è descritto nella sezione 62.3.7.

```
/*
    mcd.b
*/

define m (x, y) {
    auto n
    while (x!=y) {
        n=0
        if (x>y) {
            x=x-y
            n=1
        }
        if (n==0) {
            y=(y-x)
        }
    }
    return (x)
}

"Per calcolare il massimo comune divisore, "
"si utilizzi la funzione m (x, y): "
```

Numero primo

Il problema della determinazione se un numero sia primo o meno, è descritto nella sezione 62.3.8.

```
/*
    primo.b
*/

define p(x) {
    auto i, j
    i=2
    while (i<x) {
        scale=0
        j=(x/i)
        j=x-(j*i)
        if (j==0) {
            return (0)
        }
        i=(i+1)
    }
    return (1)
}

"Per verificare se un numero sia primo, si utilizzi la funzione p (x, y); "
"1 indica un numero primo, 0 indica un numero che non è primo. "
```

Scansione di array

In questa sezione vengono mostrati alcuni algoritmi, legati alla scansione degli array, portati in BC.

Per usare questi programmi, mancando un sistema normale di interazione con l'utilizzatore, è necessario creare un array prima di utilizzare la funzione che svolge il lavoro di ricerca o di riordino. Per esempio, nel caso della funzione '**r()**' per la ricerca sequenziale:

```
$ bc ricercaseq.b [Invio]

  Ricerca sequenziale: r (<lista>, , <elemento>, <inizio>, <fine>)
a[0]=3 [Invio]
a[1]=10 [Invio]
a[2]=33 [Invio]
a[3]=56 [Invio]
r (a[], 33, 0, 3) [Invio]

  2
[Ctrl d]
```

Ricerca sequenziale

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.

```
/*
    ricercaseq.b
*/

/* r (<lista>, <elemento>, <inizio>, <fine>) */
define r (l[], x, a, z) {
    auto i
    for (i=a; i<=z; i++) {
        if (x==l[i]) {
            return (i)
        }
    }
    /* La corrispondenza non è stata trovata. */
    return (-1)
}

"Ricerca sequenziale: r (<lista>, , <elemento>, <inizio>, <fine>) "
```

Esiste anche una soluzione ricorsiva che viene mostrata nella funzione seguente:

```
define r (l[], x, a, z) {
    if (a>z) {
        return (-1)
    }
    if (x==l[a]) {
        return (a)
    }
    return (r (l[], x, a+1, z))
}
```

Ricerca binaria

Il problema della ricerca binaria all'interno di un array, è descritto nella sezione 62.4.2.

```
/*
    ricercabin.b
*/

/* r (<lista>, <elemento>, <inizio>, <fine>) */
define r (l[], x, a, z) {
    auto m
    /* Determina l'elemento centrale. */
    scale=0
    m = ((a+z)/2)
    if (m<a) {
        /* Non restano elementi da controllare: l'elemento cercato non c'è. */
        return (-1)
    }
    if (x<l[m]) {
        /* Si ripete la ricerca nella parte inferiore. */
        return (r (l[], x, a, m-1))
    }
    if (x>l[m]) {
        /* Si ripete la ricerca nella parte superiore. */
        return (r (l[], x, m+1, z))
    }
    /* $m rappresenta l'indice dell'elemento cercato. */
    return (m)
}

"Ricerca binaria: r (<lista>, <elemento>, <inizio>, <fine>) "
```

Algoritmi tradizionali

In questa sezione vengono mostrati alcuni algoritmi tradizionali portati in BC.

Per consentire la visualizzazione del contenuto di un array è necessario predisporre una funzione apposita, che viene presentata qui, senza ripeterla nei vari esempi proposti (per evitare di visualizzare uno zero aggiuntivo, conviene assegnare il valore restituito dalla funzione stessa).

```
/* v (<lista>, <inizio>, <fine>) */
define v (l[], a, z) {
    auto j
    for (j=a; j<=z; j++) {
        (l[j])
    }
    return
}
```

Bubblesort

Il problema del Bubblesort è stato descritto nella sezione 62.5.1. Viene mostrata prima una soluzione iterativa e successivamente la funzione '**bsort()**' in versione ricorsiva.

```
/*
    bsort.b
*/

/* l[] è l'array da riordinare. */

/* b (<inizio>, <fine>) */
define b (a, z) {
    auto s, j, k

    /* Inizia il ciclo di scansione dell'array. */
    for (j=a; j<z; j++) {
        /*
            Scansione interna dell'array per collocare nella posizione j
            l'elemento giusto.
        */
        for (k=(j+1); k<=z; k++) {
            if (l[k]<l[j]) {
                /* Scambia i valori */
                s=l[k]
                l[k]=l[j]
                l[j]=s
            }
        }
    }
    return
}

"Bubblesort: l[]; t = b (<inizio>, <fine>)   "
"L'array da riordinare è l[]. "
```

Segue la funzione '**bsort()**' in versione ricorsiva:

```
define b (a, z) {
    auto s, k
    if (a<z) {
        /*
            Scansione interna dell'array per collocare nella posizione a
            l'elemento giusto.
        */
        for (k=(a+1); k<=z; k++) {
            if (l[k]<l[a]) {
                /* Scambia i valori */
                s=l[k]
                l[k]=l[a]
                l[a]=s
            }
        }
        b (a+1, z)
    }
    return
}
```

Torre di Hanoi

Il problema della torre di Hanoi è descritto nella sezione 62.5.3.

```
/*
    hanoi.b
*/

/* h (<n-anelli>, <piolo-iniziale>, <piolo-finale>) */
define h (n, i, f) {
    auto t
    if (n>0) {
        t = h (n-1, i, 6-i-f)
        "Muovi l'anello " ; n
        "dal piolo " ; i
        "al piolo " ; f
        t = h (n-1, 6-i-f, f);
    }
    return
}

"Torre di Hanoi: t = h (<n-anelli>, <piolo-iniziale>, <piolo-finale>) "
```

Quicksort

L'algoritmo del Quicksort è stato descritto nella sezione 62.5.4.

```
/*
    qsort.b
*/
```



```
/* l[] è l'array da riordinare. */

/* p (<inizio>, <fine>) */
define p (a, z) {
    auto s, i, c
    /* Si assume che a sia inferiore a z. */
    i=(a+1)
    c=z

    /* Inizia il ciclo di scansione dell'array. */
    while (1) {
        while (1) {
            /* Sposta i a destra. */
            if (l[i]>l[a]) {
                break
            }
            if (i>=c) {
              break
            }
            i=(i+1)
        }
        while (1) {
            /* Sposta c a sinistra. */
            if (l[c]<=l[a]) {
                break
            }
            c=(c-1)
        }
        if (c<=i) {
            /* È avvenuto l'incontro tra i e c. */
            break
        }
        /* Vengono scambiati i valori. */
        s=l[c]
        l[c]=l[i]
        l[i]=s

        i=(i+1)
        c=(c-1)
    }

    /*
        A questo punto l[a..z] è stata ripartita e c è la collocazione
        di l[a].
    */
    s=l[c]
    l[c]=l[a]
    l[a]=s

    /*
        A questo punto l[c] è un elemento (un valore) nella
        posizione giusta.
    */
    return (c)
}

/* q (<inizio>, <fine>) */
define q (a, z) {
    auto c
    if (z>a) {
        c = p (a, z)
        q (a, c-1)
        q (c+1, z)
    }
    return
}

"Quicksort: l[]  t = q (<inizio>, <fine>)  "
"Prima riempire l'array l[], poi chiamare la funzione q()."
```

Permutazioni

L'algoritmo ricorsivo delle permutazioni è descritto nella sezione 62.5.5.

```
/*
    permuta.b
*/

/* v (<lista>, <inizio>, <fine>) */
define v (l[], a, z) {
    auto j
    for (j=a; j<=z; j++) {
        (l[j])
    }
    return
}

/* p (<lista>, <inizio>, <fine>, <max_array>) */
define p (l[], a, z, d) {
    auto k
    auto t
    if ((z-a)>=1) {
        /*
            Inizia un ciclo di scambi tra l'ultimo elemento e uno degli
            altri contenuti nel segmento di array.
        */
        for (k=z; k>=a; k--) {
            /* Scambia i valori */
            s=l[k]
            l[k]=l[z]
            l[z]=s
```

```
            /*
                Esegue una chiamata ricorsiva per permutare un segmento
                più piccolo dell'array.
            */

            t = p (l[], a, z-1, d)

            /* Scambia i valori */
            s=l[k]
            l[k]=l[z]
            l[z]=s
        }
        return
    }
    /* Visualizza la situazione attuale dell'array. */
    " "
    t = v (l[],0,d)
    return
}

"Permutazioni: t = p (<lista>, <inizio>, <fine>, <max_array>)"
```


Parte xxxi

Basic

Basic: introduzione 1103

- Struttura fondamentale 1103
- Interprete tradizionale 1104
- Tipi di dati ed espressioni 1105
- Primi esempi banali 1107
- Strutture di controllo del flusso 1107
- Input e output 1108

Basic: esempi di programmazione 1111

- Somma tra due numeri positivi 1111
- Moltiplicazione di due numeri positivi attraverso la somma 1111
- Divisione intera tra due numeri positivi 1111
- Elevamento a potenza 1112
- Radice quadrata 1112
- Fattoriale 1112
- Ricerca sequenziale 1112

Basic: introduzione

Struttura fondamentale 1103
Numerazione delle righe 1103
Istruzioni 1103
Esecuzione di un programma 1104
Interprete tradizionale 1104
Comandi tipici dell'interprete 1104
Tipi di dati ed espressioni 1105
Espressioni numeriche 1105
Espressioni stringa 1106
Espressioni logiche 1106
Espressioni miste 1106
Array 1107
Primi esempi banali 1107
Strutture di controllo del flusso 1107
Istruzione «GOTO» 1107
Istruzione «GOSUB» 1108
Istruzione «IF» 1108
Istruzione «FOR» 1108
Istruzioni «END» e «STOP» 1108
Input e output 1108
Istruzione «PRINT» 1108
Istruzione «INPUT» o «?» 1109

Il Basic è un linguaggio di programmazione nato solo per scopi didattici, anche se ormai non si può più considerare tanto adatto neanche per questo. La semplicità di questo linguaggio fa sì che si trovino quasi sempre solo interpreti e non compilatori.

Struttura fondamentale

Di linguaggi Basic ne esistono tanti tipi, anche con estensioni che vanno molto lontano rispetto all'impostazione originale, facendone in realtà un linguaggio completamente diverso. In questa descrizione, si vuole fare riferimento al Basic tradizionale, con tutte le sue limitazioni antiche. In questo senso, l'interprete Basic per GNU/Linux che più si avvicina al livello essenziale è Bywater BASIC. [1]

Numerazione delle righe

La caratteristica tipica di un programma Basic è quella di avere le righe numerate. Infatti, non gestendo procedure e funzioni, l'unico modo per accedere a una subroutine è quello di fare riferimento alla riga in cui questa inizia. In pratica, le istruzioni iniziano con un numero di riga, progressivo, seguito da almeno uno spazio; quindi continuano con l'istruzione vera e propria:

```
110 PRINT "ciao a tutti"
120 PRINT "come va?"
```

Si può intendere che questa dipendenza dalla numerazione delle righe costituisca poi un problema per il programmatore, perché il cambiamento della numerazione implica la perdita dei riferimenti alle subroutine.

Istruzioni

Le istruzioni Basic, oltre al fatto di iniziare con il numero di riga, non hanno altre caratteristiche particolari. Generalmente utilizzano una riga e non richiedono la conclusione finale con un qualche simbolo di interpunzione.

È interessante notare invece che i commenti vanno espressi con l'istruzione '**REM**', seguita da qualcosa che poi viene ignorato, e che

le righe vuote non sono ammissibili in generale, anche se iniziano regolarmente con il numero di riga.

La natura del linguaggio Basic è tale per cui le istruzioni e i nomi delle variabili dovrebbero essere espressi sempre utilizzando le sole lettere maiuscole.

Esecuzione di un programma

L'esecuzione di un programma Basic dipende dal modo stabilito dall'interprete prescelto. L'interprete tradizionale obbliga a caricare il programma con il comando '**LOAD**' e ad avviarlo attraverso il comando '**RUN**'.

Interprete tradizionale

L'interprete Basic tradizionale è una sorta di shell che riconosce una serie di comandi interni, oltre alle istruzioni Basic vere e proprie. In pratica, attraverso l'invito di questa shell si possono eseguire singole istruzioni Basic, oppure comandi utili a gestire il file di un programma completo. Per esempio, avviando il Bywater BASIC, si ottiene quanto segue:

```
$ bwbasic [Invio]

bwBASIC:
```

In pratica, '**bwBASIC:**' rappresenta l'invito. L'esempio seguente mostra l'inserimento di alcune istruzioni Basic, allo scopo di eseguire la moltiplicazione 6*7.

```
bwBASIC: A=6 [Invio]

bwBASIC: B=7 [Invio]

bwBASIC: C=A*B [Invio]

bwBASIC: PRINT C [Invio]

 42
```

Comandi tipici dell'interprete

L'interprete Basic tipico mette a disposizione alcuni comandi, che risultano essenziali per la gestione di un programma Basic.

Opzione	Descrizione
`LIST [`*riga_iniziale*`[-`*riga_finale*`][,...]`	Elenca le righe del programma selezionate dagli intervalli indicati come argomento. Se non viene indicato alcun argomento, la visualizzazione viene fatta a partire dalla prima riga; se viene indicata solo la riga iniziale, la visualizzazione riguarda esclusivamente quella riga. L'esempio seguente serve a visualizzare la riga 100 e poi l'intervallo da 150 a 200: '**LIST 100, 150-200**'
`RUN [`*riga_iniziale*`]`	Il comando '**RUN**' viene usato normalmente senza argomenti, per avviare il programma caricato nell'interprete. Se si aggiunge il numero di una riga, quel punto viene utilizzato per iniziare l'interpretazione ed esecuzione del programma.
`NEW`	Cancella il programma che eventualmente fosse caricato nell'interprete.
`LOAD` *file*	Carica il programma indicato dal nome del file posto come argomento. Se esiste già un programma in memoria, quello viene eliminato. Solitamente, il nome del file deve essere indicato delimitandolo tra apici doppi. È probabile che l'interprete aggiunga un'estensione predefinita od obbligatoria.
`SAVE` *file*	Salva il programma con il nome specificato come argomento. Solitamente, il nome del file deve essere indicato delimitandolo tra apici doppi. È probabile che l'interprete aggiunga un'estensione predefinita od obbligatoria.

Opzione	Descrizione
`DEL` *riga_iniziale*`[-`*riga_finale*`][,...]`	Elimina le righe indicate dall'argomento. Può trattarsi di una sola riga, o di un intervallo, o di una serie di intervalli.
`RENUM [`*riga_iniziale*`[,`*incremento*`]]`	Rinumera le righe del programma, aggiornando i riferimenti alle subroutine. È possibile indicare il numero iniziale e anche l'incremento. Di solito, se non viene specificato alcun argomento, la riga iniziale ha il numero 10 e l'incremento è sempre di 10.
`BYE` `QUIT`	Termina il funzionamento dell'interprete Basic.

L'inserimento delle righe di programma attraverso l'interprete Basic, avviene iniziando le istruzioni con il numero di riga in cui queste devono essere collocate. Ciò permette così di inserire righe aggiuntive anche all'interno del programma. Se si utilizzano numeri di righe già esistenti, queste vengono sostituite.

Quando un'istruzione Basic viene inserita senza il numero iniziale, questa viene eseguita immediatamente.

Tipi di dati ed espressioni

I tipi di dati gestibili in Basic sono generalmente solo i numeri reali (numeri a virgola mobile con approssimazione che varia a seconda dell'interprete) e le stringhe.

I numeri vengono indicati senza l'uso di delimitatori; se necessario, è possibile rappresentare valori decimali con l'uso del punto di separazione; inoltre è generalmente ammissibile la notazione esponenziale. L'esempio seguente mostra due modi di rappresentare lo stesso numero.

```
123.456
1.23456E+2
```

Le stringhe si rappresentano delimitandole attraverso apici doppi (possono essere ammessi anche gli apici singoli, ma questo dipende dall'interprete) e sono soggette a un limite di dimensione che dipende dall'interprete (spesso si tratta di soli 255 caratteri).

Le variabili sono distinte in base al fatto che servano a contenere numeri o stringhe. Per la precisione, le variabili che contengono stringhe, hanno un nome che termina con il simbolo dollaro ('**$**'). I nomi delle variabili, a parte l'eventuale aggiunta del dollaro per le stringhe, sono soggetti a regole differenti a seconda dell'interprete; in particolare occorre fare attenzione al fatto che l'interprete potrebbe distinguere tra maiuscole e minuscole. In origine, si poteva utilizzare una sola lettera alfabetica!

L'assegnamento di una variabile avviene attraverso l'operatore '**=**', secondo la sintassi seguente:

```
[LET] variabile=valore
```

L'uso esplicito dell'istruzione '**LET**' è facoltativo.

Espressioni numeriche

Gli operatori tipici che intervengono su valori numerici, restituendo valori numerici, sono elencati nella tabella u133.6.

Tabella u133.6. Elenco degli operatori utilizzabili in presenza di valori numerici, all'interno di espressioni numeriche. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
var = *valore*	Assegna alla variabile il valore alla destra.
- *op1*	Inverte il segno dell'operando.
op1 + *op2*	Somma i due operandi.

Operatore e operandi	Descrizione
op1 - *op2*	Sottrae dal primo il secondo operando.
op1 * *op2*	Moltiplica i due operandi.
op1 / *op2*	Divide il primo operando per il secondo.
op1 MOD *op2*	Modulo: il resto della divisione tra il primo e il secondo operando.
op1 ^ *op2*	Eleva il primo operando alla potenza del secondo.
SQRT *op1*	Calcola la radice quadrata dell'operando.
SIN *op1*	Calcola il seno dell'operando.
COS *op1*	Calcola il coseno dell'operando.
TAN *op1*	Calcola la tangente dell'operando.
ARCTAN *op1*	Calcola l'arcotangente dell'operando.
LOG *op1*	Calcola il logaritmo naturale dell'operando.
ABS *op1*	Calcola il valore assoluto dell'operando.

Le parentesi tonde possono essere utilizzate per indicare esplicitamente l'ordine dell'elaborazione delle espressioni.

Espressioni stringa

L'unico tipo di espressione che restituisce una stringa a partire da stringhe, è il concatenamento che si ottiene con l'operatore '**+**':

```
stringa_1+stringa_2
```

Espressioni logiche

Le espressioni logiche si possono realizzare a partire da dati numerici, da dati stringa e dal risultato di altre espressioni logiche. La tabella u133.7 mostra gli operatori fondamentali.

Tabella u133.7. Elenco degli operatori utilizzabili nelle espressioni logiche. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 = *op2*	I due numeri, o le due stringhe sono uguali.
op1 < *op2*	Il primo operando è minore del secondo.
op1 > *op2*	Il primo operando è maggiore del secondo.
op1 <= *op2*	Il primo operando è minore o uguale al secondo.
op1 >= *op2*	Il primo operando è maggiore o uguale al secondo.
op1 <> *op2*	I due operandi sono diversi.
cond1 AND *cond2*	Le due condizioni sono entrambe vere.
cond1 OR *cond2*	Almeno una delle due condizioni è vera.
NOT *cond1*	Inverte il risultato logico della condizione.

Espressioni miste

Alcuni operatori utilizzano valori di tipo diverso dal tipo di dati che restituiscono. La tabella u133.8 mostra alcuni di questi.

Tabella u133.8. Elenco di altri operatori.

Operatore e operandi	Descrizione
VAL *stringa*	Valuta la stringa trattandola come un'espressione numerica.
LEN *stringa*	Restituisce la lunghezza della stringa in caratteri.
STR$ *numero*	Restituisce una stringa contenente il numero indicato come argomento.

Array

Gli array in Basic possono essere a una o più dimensioni, a seconda dell'interprete. In ogni caso, dovrebbero essere distinti in base al contenuto: solo numeri o solo stringhe. L'indice del primo elemento dovrebbe essere zero. La dichiarazione avviene nel modo seguente:

```
DIM nome(dimensione_1[, dimensione_2]...)
```

```
DIM nome$(dimensione_1[, dimensione_2]...)
```

Nel primo caso si tratta di un array con elementi numerici, nel secondo si tratta di un array con elementi stringa.

Primi esempi banali

L'esempio seguente è il più banale, emette semplicemente la stringa '**"Ciao Mondo!"**' attraverso lo standard output:

```
10 print "Ciao Mondo!"
```

Per eseguire il programma basta utilizzare il comando '**RUN**':

```
RUN [Invio]

 Ciao Mondo!
```

L'esempio seguente genera lo stesso risultato di quello precedente, ma con l'uso di variabili:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 PRINT A$; " "; B$
```

L'esempio seguente genera lo stesso risultato di quello precedente, ma con l'uso del concatenamento di stringa:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 PRINT A$+" "+B$
```

L'esempio seguente mostra l'uso di una costante e di una variabile numerica:

```
10 A$ = "Ciao"
20 B$ = "Mondo"
30 N = 1000
40 PRINT N; "volte "; A$; " "; B$
```

Il risultato che si ottiene dovrebbe essere il seguente:

```
 1000 volte Ciao Mondo!
```

Strutture di controllo del flusso

Il Basic è un linguaggio di programmazione molto povero dal punto di vista delle strutture di controllo. In modo particolare sono assenti funzioni e procedure. Per fare riferimenti a porzioni di codice occorre sempre indicare un numero di riga, attraverso le istruzioni '**GOTO**' o '**GOSUB**'.

Istruzione «GOTO»

```
GOTO riga
```

Si tratta dell'istruzione di salto incondizionato e senza ritorno. In pratica, l'esecuzione del programma prosegue dalla riga indicata come argomento, perdendo ogni riferimento al punto di origine.

Istruzione «GOSUB»

```
GOSUB riga
```

Si tratta dell'istruzione di salto incondizionato con ritorno. L'esecuzione del programma prosegue dalla riga indicata come argomento e, quando poi viene incontrata l'istruzione '**RETURN**', il programma riprende dalla riga successiva a quella in cui è avvenuta la chiamata. Questo è l'unico modo offerto dal Basic tradizionale per la realizzazione di subroutine

L'esempio seguente mostra un programma completo che visualizza il messaggio '**"Ciao"**' e poi il messaggio '**"Mondo"**':

```
10 GOTO 50
20 A$ = "Ciao"
30 PRINT A$
40 RETURN
50 GOSUB 20
60 B$ = "Mondo"
70 PRINT B$
```

Istruzione «IF»

```
IF condizione THEN istruzione [ELSE istruzione]
```

Se la condizione si verifica, viene eseguita l'istruzione posta dopo la parola chiave '**THEN**', altrimenti, se esiste, quella posta dopo la parola chiave '**ELSE**'. La situazione è tale per cui le istruzioni condizionate sono prevalentemente '**GOTO**' e '**GOSUB**'.

L'esempio seguente emette la stringa '**"Ottimo"**' se la variabile '**N**' contiene un valore superiore a 100; altrimenti esegue la subroutine che inizia a partire dalla riga 50.

```
150 IF N > 100 THEN PRINT "Ottimo" ELSE GOSUB 50
```

Istruzione «FOR»

```
FOR variabile_num = inizio TO fine [STEP incremento]
istruzioni
…
NEXT
```

Esegue le istruzioni e ogni volta incrementa la variabile numerica indicata, assegnandole inizialmente il valore posto dopo il simbolo '**=**'. Il blocco di istruzioni viene eseguito fino a quando la variabile raggiunge il valore finale stabilito; l'incremento è unitario, a meno che sia stato indicato diversamente attraverso l'argomento della parola chiave '**STEP**'.

Istruzioni «END» e «STOP»

La conclusione, o l'interruzione del programma può essere indicata esplicitamente utilizzando l'istruzione '**END**' oppure l'istruzione '**STOP**'. La prima corrisponde all'interruzione dovuta a una conclusione normale, la seconda serve a generare un messaggio di errore e si presta per l'interruzione del programma in presenza di situazioni anomale.

Input e output

L'input e l'output del Basic tradizionale è molto povero, riguardando prevalentemente l'acquisizione di dati da tastiera e l'emissione di testo sullo schermo.

Istruzione «PRINT»

```
PRINT operando [{, | ;}…]
```

L'istruzione '**PRINT**' permette di emettere sullo schermo una stringa corrispondente agli operandi utilizzati come argomenti. Eventuali valori numerici vengono convertiti in stringhe automaticamente. Gli operandi possono essere elencati utilizzando la virgola o il punto e virgola. Gli esempi seguenti sono equivalenti:


```
10 PRINT 1234, "saluti"
```

```
10 PRINT 1234; "saluti"
```

```
10 A = 1234
20 PRINT A; "saluti"
```

```
10 A = 1234
20 M$ = "saluti"
30 PRINT A; M$
```

Se come operando si vuole utilizzare il risultato di un'espressione, di qualunque tipo, può essere necessario l'uso di parentesi tonde, come nell'esempio seguente, in cui si vuole emettere il risultato del coseno di zero:

```
10 PRINT ( COS 0 )
```

Istruzione «INPUT» o «?»

```
INPUT [invito ;] variabile [, variabile]…
```

```
? [invito ;] variabile [, variabile]…
```

Attraverso questa istruzione è possibile inserire un valore in una variabile, o una serie di valori in una serie di variabili. Se viene indicata la stringa dell'invito, questa viene visualizzata prima di attendere l'inserimento da parte dell'utente; altrimenti viene visualizzato semplicemente un punto interrogativo.

Se si indica un elenco di variabili, queste devono essere dello stesso tipo (tutte numeriche o tutte stringa) e il loro inserimento viene atteso in modo sequenziale da parte dell'utente.

L'esempio seguente rappresenta l'inserimento di una stringa senza invito e di una coppia di numeri con invito:

```
10 INPUT A$
20 INPUT "Inserisci la coppia di numeri "; X, Y
```

[1] **Bywater BASIC** GNU GPL

Basic: esempi di programmazione

Somma tra due numeri positivi 1111
Moltiplicazione di due numeri positivi attraverso la somma . 1111
Divisione intera tra due numeri positivi 1111
Elevamento a potenza 1112
Radice quadrata 1112
Fattoriale 1112
Ricerca sequenziale 1112

In questo capitolo si raccolgono solo alcuni esempi molto semplici di programmazione in Basic. Infatti, questo linguaggio di programmazione non si presta per la rappresentazione di algoritmi complessi.

Somma tra due numeri positivi 1111
Moltiplicazione di due numeri positivi attraverso la somma . 1111
Divisione intera tra due numeri positivi 1111
Elevamento a potenza 1112
Radice quadrata 1112
Fattoriale 1112
Ricerca sequenziale 1112

Somma tra due numeri positivi

Il problema della somma tra due numeri positivi, attraverso l'incremento unitario, è descritto nella sezione 62.3.1.

```
1000 REM ============================================================
1010 REM somma.bas
1020 REM Somma esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = X
1080 FOR I = 1 TO Y
1090 LET Z = Z + 1
1100 NEXT
1110 PRINT X; "+"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Moltiplicazione di due numeri positivi attraverso la somma

Il problema della moltiplicazione tra due numeri positivi, attraverso la somma, è descritto nella sezione 62.3.2.

```
1000 REM ============================================================
1010 REM moltiplica.bas
1020 REM Moltiplica esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 0
1080 FOR I = 1 TO Y
1090 LET Z = Z + X
1100 NEXT
1110 PRINT X; "*"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Divisione intera tra due numeri positivi

Il problema della divisione tra due numeri positivi, attraverso la sottrazione, è descritto nella sezione 62.3.3.

```
1000 REM ============================================================
1010 REM dividi.bas
1020 REM Divide esclusivamente valori positivi.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 0
1080 LET I = X
1090 IF I < Y THEN GOTO 1130
1100 LET I = I - Y
1110 LET Z = Z + 1
1120 GOTO 1090
1130 PRINT X; "/"; Y; "="; Z
1140 END
1150 REM ============================================================
```

Elevamento a potenza

«

Il problema dell'elevamento a potenza tra due numeri positivi, attraverso la moltiplicazione, è descritto nella sezione 62.3.4.

```
1000 REM ============================================================
1010 REM exp.bas
1020 REM Eleva a potenza.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il primo valore "; X
1060 INPUT "Inserisci il secondo valore "; Y
1070 LET Z = 1
1080 FOR I = 1 TO Y
1090 LET Z = Z * X
1100 NEXT
1110 PRINT X; "^"; Y; "="; Z
1120 END
1130 REM ============================================================
```

Radice quadrata

«

Il problema della radice quadrata è descritto nella sezione 62.3.5.

```
1000 REM ============================================================
1010 REM radice.bas
1020 REM Radice quadrata intera.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il valore "; X
1060 LET Z = 0
1070 LET T = 0
1080 REM Inizio del ciclo di calcolo
1090 LET T = Z * Z
1100 IF T > X THEN GOTO 1130
1110 LET Z = Z + 1
1120 GOTO 1080
1130 REM Riprende il flusso normale
1140 LET Z = Z - 1
1150 PRINT "radq("; X; ") ="; Z
1160 END
1170 REM ============================================================
```

Fattoriale

«

Il problema del fattoriale è descritto nella sezione 62.3.6.

```
1000 REM ============================================================
1010 REM fatt.bas
1020 REM Fattoriale.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il valore "; X
1060 LET Z = X
1070 FOR I = (X - 1) TO 1 STEP -1
1080 LET Z = Z * I
1090 NEXT
1100 PRINT "fatt("; X; ") ="; Z
1110 END
1120 REM ============================================================
```

Ricerca sequenziale

«

Il problema della ricerca sequenziale all'interno di un array, è descritto nella sezione 62.4.1.


```
1000 REM ============================================================
1010 REM ricercaseq.bas
1020 REM Ricerca sequenziale.
1030 REM ============================================================
1040 REM
1050 INPUT "Inserisci il numero di elementi "; N
1060 DIM A(N)
1070 FOR I = 0 TO N-1
1080 PRINT "A("; I; ") ="
1090 INPUT A(I)
1100 NEXT
1110 INPUT "Inserisci il valore da cercare "; X
1120 FOR I = 0 TO N-1
1130 IF X = A(I) THEN GOTO 1170
1140 NEXT
1160 GOTO 1190
1170 PRINT "L'elemento A("; I; ") contiene il valore "; X
1180 END
1190 PRINT "Il valore "; X; " non è stato trovato"
1200 END
1210 REM ============================================================
```

File «.DBF»: dBase III e derivati 1117
Dbview 1117
nanoBase 1997 1121
What is it 1121
The dot command line 1121
The menu 1121
The macro recording, compiling and execution 1122
The report system 1123
The integrated text editor 1123
The internal documentation 1124
Download it 1124
Bugs and known problems 1124
nanoBase 1997 user manual 1127
Dos xBase 1134
Composition 1137
How to use nB 1139
Status line 1140
The dot line 1140
The menu system 1140
The text editor DOC() 1149
The help text file 1150
Macro 1150
Data types 1154
Operators 1160
Delimiters 1161
Code blocks 1162
Standard functions 1162
nB functions 1208
Normal command substitution 1237
nB command substitution functions 1247
RPT: the nB print function 1251
How can I... 1254
The source files 1255
Clean the Clipper 5.2 1257
Step 1: try to compile with the /P parameter 1259
Step 2: understand well the use of code blocks 1259
Step 3: understand the object programming 1260
Step 4: understand the get object 1261
Step 5: trying to stop using commands 1262
Step 6: free yourself from STD.CH - /U 1277
Step 7: take control over all include files 1277

File «.DBF»: dBase III e derivati

Dbview 1117
DBF2pg 1118

Il software basato sui file in formato '`.DBF`', ovvero quelli di dBase III, negli anni 1980 è stato molto importante nell'ambito del sistema operativo Dos. Nel suo piccolo ha permesso agli utenti di quel sistema operativo di realizzare delle strutture di dati che si avvicinavano alle potenzialità di una base di dati relazionale.

Ancora oggi si trovano programmi applicativi gestionali basati su questo formato, scritti probabilmente con il famoso compilatore Clipper. Attualmente è disponibile il compilatore Harbour, che si ripromette di offrire un ambiente totalmente compatibile con il passato; tuttavia è possibile leggere il contenuto di questi file attraverso alcuni piccoli programmi.

Dbview

Il programma '**`dbview`**' [1] consente di leggere il contenuto dei file '`.DBF`' di dBase III e probabilmente anche le versioni di dBase IV.

`dbview` [***opzioni***] ***file_dbf***

Se viene avviato senza opzioni, si ottiene la visualizzazione del contenuto del file indicato nel formato predefinito, come si vede dall'esempio seguente:

```
Articolo   : 1
Descr      : bicicletta uomo
Prezzo u   : 500.00
Import     : T
Scadenza   : 20011120
Note       : 2

Articolo   : 2
Descr      : bicicletta donna
Prezzo u   : 550.00
Import     :
Scadenza   : 20011120
Note       : 3

Articolo   : 3
Descr      : bicicletta uomo/donna leggera
Prezzo u   : 600.00
Import     :
Scadenza   : 20011120
Note       : 4
```

In realtà, così facendo, i nomi degli attributi vengono mostrati in modo diverso dal reale, utilizzando anche le lettere minuscole ed eliminando i trattini bassi. Utilizzando l'opzione '**`-r`**', la prima tupla apparirebbe così:

```
ARTICOLO   : 1
DESCR      : bicicletta uomo
PREZZO_U   : 500.00
IMPORT     : T
SCADENZA   : 20011111
NOTE       : 2
```

È necessario osservare che gli attributi booleani (in questo caso si tratta di quello intitolato '**`IMPORT`**') mostrano solo la lettera '**`T`**' per il valore *Vero*, altrimenti non si ha alcuna indicazione; inoltre, le date vengono espresse secondo il formato ***aaaammgg***. Infine, dall'esempio non si intuisce, ma l'attributo '**`NOTE`**' è di tipo «memo» e in questo caso si sono persi i dati.

I dati contenuti nei file '`.DBF`', dal momento che sono stati memorizzati presumibilmente con un sistema operativo Dos, utilizzano molto probabilmente un insieme di caratteri ristretto e incompatibile con gli standard comuni; pertanto, è probabile che sia necessario rielaborare ciò che si ottiene con '**`dbview`**' attraverso un programma di conversione come Recode (sezione 47.8.1). Tuttavia, è bene considerare che nella storia dei file '`.DBF`' sono state usate anche codifiche differenti dal solito IBM437 e di questo occorre tenerne conto quando ci si accorge che la conversione non funziona come ci si aspetterebbe.

Tabella u135.3. Alcune opzioni.

Opzione	Descrizione
`--browse` `-b`	Se si utilizza questa opzione, le tuple vengono mostrate su una sola riga per volta, separando gli attributi con un simbolo, il separatore, che di solito è costituito dai due punti (‘`:`’).
`--delimiter` ***x*** `-d` ***x***	Con questa opzione è possibile specificare il simbolo da utilizzare per separare gli attributi delle tuple che vengono visualizzate. Il simbolo di separazione predefinito sono i due punti (‘`:`’)
`--description` `-e` ***x***	In questo caso, oltre a mostrare il contenuto del file, nella parte iniziale vengono riepilogate le caratteristiche degli attributi contenuti.
`--omit` `-o` ***x***	Non elenca il contenuto del file, ma si limita a dare le altre informazioni se richieste attraverso le opzioni opportune.
`--reserve` `-r` ***x***	Mostra i nomi degli attributi così come sono stati memorizzati.

Segue la descrizione di alcuni esempi.

- `$` **`dbview articoli.dbf`** [*Invio*]

  Elenca il contenuto del file ‘`articoli.dbf`’ nella forma predefinita.

- `$` **`dbview -b articoli.dbf`** [*Invio*]

  Mostra le tuple utilizzando una sola riga per ognuna.

- `$` **`dbview -b articoli.dbf | recode ibm437:latin1`** [*Invio*]

  Come nell’esempio precedente, ma utilizza ‘**`recode`**’ per trasformare i caratteri speciali che altrimenti non sarebbero visibili correttamente (per esempio le lettere accentate).

DBF2pg

Il programma ‘**`dbf2pg`**’ [2] consente di leggere il contenuto di un file ‘`.DBF`’ e di inserire i dati relativi in una relazione di una base di dati di PostgreSQL.

`dbf2pg` [***opzioni***] ***file_dbf***

In base alle opzioni che vengono indicate, i dati possono essere aggiunti a una relazione esistente, oppure possono sostituire le tuple di tale relazione, oppure si può creare una relazione da zero. Quello che conta è che i permessi fissati attraverso PostgreSQL consentano l’accesso e le operazioni che si intendono svolgere.

‘**`dbf2pg`**’ non è in grado di trasferire gli attributi «memo», quelli che tradizionalmente venivano creati utilizzando file con estensione ‘`.DBT`’.

Tabella u135.4. Alcune opzioni.

Opzione	Descrizione
`-v` `-vv`	Permette di avere informazioni sulle operazioni svolte, ottenendo un dettaglio maggiore nel secondo caso.
`-h` ***nodo***	Permette di specificare il nodo a cui accedere per connettersi con il servente di PostgreSQL. In mancanza di questa indicazione, viene tentato l’accesso a *localhost*.
`-d` ***base_di_dati***	Permette di specificare il nome della base di dati a cui ci si vuole connettere. In mancanza di questa indicazione, viene tentata la connessione con la base di dati ‘**`test`**’.

Opzione	Descrizione
`-t` ***relazione***	Permette di specificare il nome della relazione in cui si vogliono trasferire i dati del file ‘`.DBF`’. In mancanza di questa indicazione, viene tentato l’inserimento nella relazione ‘**`test`**’.
`-D`	Con questa opzione, si fa in modo di cancellare il contenuto della relazione di destinazione, prima di iniziare l’inserimento dei dati.
`-c`	Richiede espressamente che sia creata la relazione di destinazione. In mancanza di questa opzione, la relazione deve essere già disponibile, altrimenti l’operazione fallisce. Nel caso si utilizzi questa opzione mentre una relazione con lo stesso nome esiste già, si ottiene la cancellazione del suo contenuto prima di iniziare, come se fosse stata usata al suo posto l’opzione ‘**`-D`**’.
`-f`	Prima di procedere, converte i nomi degli attributi in modo che questi siano scritti utilizzando solo lettere minuscole.
`-l` `-u`	Con l’opzione ‘**`-l`**’ si fa in modo che il contenuto degli attributi venga convertito in lettere minuscole, mentre con l’opzione ‘**`-u`**’ si ottiene una conversione in maiuscole.
`-s` ***nome_vecchio=nome_nuovo*** [, ***nome_vecchio=nome_nuovo***] …	Con questa opzione si può stabilire la sostituzione di alcuni nomi degli attributi della relazione. Ciò può essere particolarmente utile nel caso in cui i nomi originali siano incompatibili con PostgreSQL.
`-s` ***n_riga_iniziale*** `-e` ***n_riga_finale***	Le opzioni ‘**`-s`**’ e ‘**`-e`**’ permettono di definire l’intervallo di righe da trasferire, dove nel primo caso si indica la riga iniziale e nel secondo quella finale. Se non si indicano, il trasferimento parte dall’inizio e prosegue fino alla fine.

Segue la descrizione di alcuni esempi.

- `$` **`dbf2pg -d Anagrafe -c -t Indirizzi address.dbf`** [*Invio*]

  Crea la relazione ‘**`Indirizzi`**’ nella base di dati ‘**`Anagrafe`**’ disponibile presso l’elaboratore locale, prelevando i dati dal file ‘`address.dbf`’.

- `$` **`dbf2pg -h localhost -d Anagrafe -c -t Indirizzi address.dbf`** [*Invio*]

  Esattamente come nell’esempio precedente, con l’indicazione precisa del nodo locale.

[1] **Dbview** GNU GPL

[2] **DBF2pg** software libero con licenza speciale

nanoBase 1997

What is it 1121
The dot command line 1121
The menu 1121
The macro recording, compiling and execution 1122
The report system 1123
The integrated text editor 1123
The internal documentation 1124
Download it 1124
Bugs and known problems 1124

An old, but free xBase for Dos.[1]

What is it

nanoBase [2] is a Dos program that works essentially as:

- a dot command line xBase,
- a menu driven xBase,
- a xBase program interpreter.

nanoBase 1997 is compiled in two versions: a small one to be used with old computers (x86-16 with 640 Kibyte RAM), and a second one to be used with better computers, at least i286 (or better) with 2 Mibyte RAM.

The dot command line

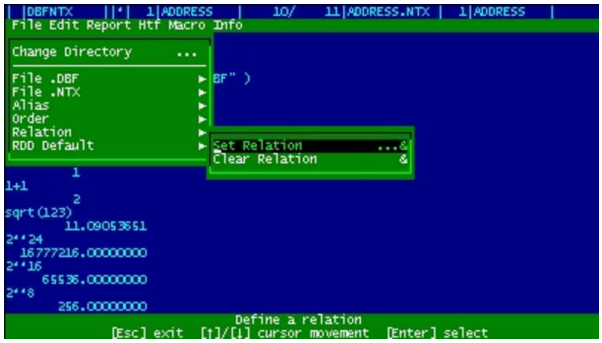
Figure u136.1. The dot line.

The dot command line is the first face of nanoBase, the one that appears starting the program normally. It recalls the dot line command of the old xBases.

Please note that **nanoBase recognise only expressions** (that is: no commands).

The menu

Figure u136.2. The file menu.



Pressing [*F10*] the nanoBase menu appears.

From this menu the operations are easier than writing all commands on a prompt line, but it is always possible to come back to the dot line to do an operation not available from the menu.

The macro recording, compiling and execution

Figure u136.3. The macro menu.

nanoBase is able to record some actions made with the menu and all what is correctly typed from the dot prompt. This may be the begin for a little program (called macro inside nanoBase) that can be executed as it is (ASCII), or compiled into another format, faster to execute.

Macros for nanoBase are made with a reduced set of the Clipper syntax. The statements recognised from nanoBase are:

```
PROCEDURE procedure_name
    statements...
    [RETURN]
    statements...
ENDPROCEDURE
```

```
DO PROCEDURE procedure_name
BEGIN SEQUENCE
    statements...
    [BREAK]
    statements...
END
```

```
DO CASE
CASE lCondition1
    statements...
[CASE lCondition2]
    statements...
[OTHERWISE]
    statements...
END
```

```
WHILE lCondition
    statements...
    [EXIT]
    statements...
    [LOOP]
    statements...
END
```

```
IF lCondition1
    statements...
    [ELSE]
    statements...
END
```


- the '**FOR**' loop is not available (too difficult to implement),
- there may be no user defined functions (code blocks may be created instead),
- procedure calls cannot transfer variables,
- there are only public (global) variables.

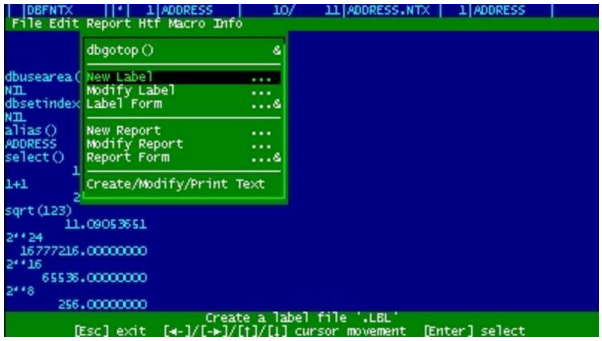
Beside these limitations, there are many added functions to the standard language that make the programming easier.

All you need is inside '**NB.EXE**':

- the utility to handle manually the data,
- the macro compiler,
- the macro executor.

The report system

Figure u136.4. The report menu.



nanoBase can handle label ('.LBL') and form ('.FRM') files in the dBaseIII format. Labels and forms may be created and edited inside nanoBase. Beside these old report system there is another way to make a little bit complicated reports without making a complex macro: it is called RPT.

A RPT file is a ASCII file with text mixed with code. The text may contain variables (usually a field or an expression containing fields).

To make a complex report some work is needed, but surely less than the time needed to make a report program.

The main purpose of it was to be able to print text with variables (typically names and addresses) for every record of a particular '.DBF' file. Now the RPT system makes something more.

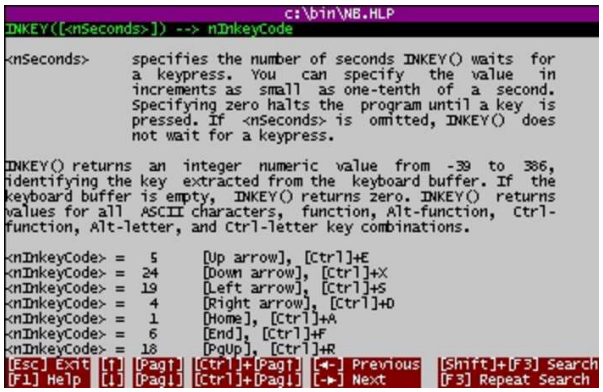
The integrated text editor

Figure u136.5. The integrated text editor.

nanoBase contains an integrated text editor not particularly good, but very usefull for RPT files (as the expression insertion is very easy with the use of the [*F2*] key) and whenever there isn't any other editor there.

The internal documentation

Figure u136.6. The internal documentation.



nanoBase's documentation si translated also inside the HTF format: '`NB.HLP`'. Pressing [*F1*], normally, a contextual piece of the manual appears.

Some standard functions have its own internal help, contained inside the '`.EXE`' file. This was made to help programming with nanoBase.

Download it

Here is the 1997 edition of nanoBase.

- EXE for small computers.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a1.zip
  http://www.google.com/search?q=nbase7a1.zip
- EXE for i286 with more than 2 Mibyte.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a2.zip
  http://www.google.com/search?q=nbase7a2.zip
- Runtime for small computers.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a3.zip
  http://www.google.com/search?q=nbase7a3.zip
- Documentation in many different formats.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a4.zip
  http://www.google.com/search?q=nbase7a4.zip
- Macro programming examples.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a5.zip
  http://www.google.com/search?q=nbase7a5.zip
- Source for version 96.06.16, without mouse support (1996).
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a6.zip
  http://www.google.com/search?q=nbase7a6.zip
- Source for version 1997.
  ftp://ftp.simtel.net/pub/simtelnet/msdos/database/nbase7a7.zip
  http://www.google.com/search?q=nbase7a7.zip

Bugs and known problems

Here is the list of known bugs and problems.

- Comparison with floating point numbers may fail. It is better to convert numbers into string before comparing them.
- Macros may be contained inside ASCII files or a ''compiled'' '`.DBF`' file. In the second case, when nanoBase executes the macro, a work area (the last available one) is used, so it should not be closed or the macro execution will be stopped. A '**`dbcloseall()`**' will stop execution of the macro. In substitution of '**`dbcloseall()`**', '**`DBCLOSE()`**' should be used.

- To simplify the macro interpretation, lines such as this:

```
qqout( "You can't do that // you can't do that!" )
```

  will generate an error as the interpreter will read only:

```
qqout( "You can't do that
```

- nanoBase works good also if you have a screen configuration that permits you to show more than the usual 80 columns and 25 lines, but the library used to handle the mouse is not able to work outside the 80×25 area.

[1] This material appeared originally at '`http://www.geocities.com/SiliconValley/7737/nanobase.html`', in 1997.

[2] **nanoBase** GNU GPL

Dos xBase 1134
.DBF files 1134
Index files 1135
Relations 1136
Composition 1137
How to use nB 1139
Status line 1140
The dot line 1140
The menu system 1140
Menu File 1141
Menu Edit 1144
Menu Report 1145
Menu HTF 1148
Menu Macro 1148
Menu Info 1149
Menu Doc 1149
The text editor DOC() 1149
The help text file 1150
Macro 1150
Macro statements 1151
Variable declaration 1152
Macro structure 1152
Macro comments 1153
Macro long lines split 1153
The macro recorder 1153
Data types 1154
Character 1154
Memo 1155
Date 1155
Numeric 1156
Logical 1157
NIL 1157
Array 1158
Code block 1159
Operators 1160
Delimiters 1161
Code blocks 1162
Standard functions 1162
AADD() 1162
ABS() 1162
ACLONE() 1162
ACOPY() 1163
ADEL() 1163
AEVAL() 1163
AFILL() 1163
AINS() 1164
ALERT() 1164
ALIAS() 1164
ALLTRIM() 1164
ARRAY() 1165
ASC() 1165
ASCAN() 1165
ASIZE() 1165

ASORT() 1165
AT() 1166
ATAIL() 1166
BIN2I() 1166
BIN2L() 1166
BIN2W() 1167
BOF() 1167
CDOW() 1167
CHR() 1167
CMONTH() 1167
COL() 1167
COLORSELECT() 1168
CTOD() 1168
CURDIR() 1168
DATE() 1168
DAY() 1168
DBAPPEND() 1168
DBCLEARFILTER() 1169
DBCLEARINDEX() 1169
DBCLEARRELATION() 1169
DBCLOSEALL() 1169
DBCLOSEAREA() 1169
DBCOMMIT() 1169
DBCOMMITALL() 1169
DBCREATE() 1169
DBCREATEINDEX() 1170
DBDELETE() 1170
DBEVAL() 1170
DBFILTER() 1171
DBGOBOTTOM() 1171
DBGOTO() 1171
DBGOTOP() 1171
DBRECALL() 1171
DBREINDEX() 1171
DBRELATION() 1171
DBRLOCK() 1171
DBRLOCKLIST() 1172
DBRSELECT() 1172
DBRUNLOCK() 1172
DBSEEK() 1172
DBSELECTAREA() 1172
DBSETDRIVER() 1172
DBSETFILTER() 1173
DBSETINDEX() 1173
DBSETORDER() 1173
DBSETRELATION() 1173
DBSKIP() 1174
DBSTRUCT() 1174
DBUNLOCK() 1174
DBUNLOCKALL() 1174
DBUSEAREA() 1174
DBDELETE() 1175
DESCEND() 1175
DEVOUT() 1175
DEVOUTPICT() 1175
DEVPOS() 1176
DIRECTORY() 1176
DISKSPACE() 1176

DISPBOX() 1177
DISPOUT() 1177
DOW() 1177
DTOC() 1177
DTOS() 1177
EMPTY() 1178
EOF() 1178
EVAL() 1178
EXP() 1178
FCLOSE() 1178
FCOUNT() 1179
FCREATE() 1179
FERASE() 1179
FERROR() 1179
FIELDBLOCK() 1180
FIELDGET() 1180
FIELDNAME() 1180
FIELDPOS() 1180
FIELDPUT() 1180
FIELDWBLOCK() 1180
FILE() 1181
FLOCK() 1181
FOPEN() 1181
FOUND() 1181
FREAD() 1181
FREADSTR() 1182
FRENAME() 1182
FSEEK() 1182
FWRITE() 1183
GETENV() 1183
HARDCR() 1183
HEADER() 1183
I2BIN() 1183
IF() 1184
INDEXEXT() 1184
INDEXKEY() 1184
INDEXORD() 1184
INKEY() 1184
INT() 1187
ISALPHA() 1187
ISCOLOR() 1187
ISDIGIT() 1187
ISLOWER() 1188
ISPRINTER() 1188
ISUPPER() 1188
L2BIN() 1188
LASTKEY() 1188
LASTREC() 1188
LEFT() 1188
LEN() 1189
LOG() 1189
LOWER() 1189
LTRIM() 1189
LUPDATE() 1189
MAX() 1189
MAXCOL() 1189
MAXROW() 1190
MEMOEDIT() 1190

MEMOLINE() ... 1191
MEMOREAD() ... 1191
MEMORY() ... 1191
MEMOTRAN() ... 1192
MEMOWRIT() ... 1192
MEMVARBLOCK() ... 1192
MIN() ... 1192
MLCOUNT() ... 1193
MLCTOPOS() ... 1193
MLPOS() ... 1193
MONTH() ... 1193
MPOSTOLC() ... 1194
NETERR() ... 1194
NETNAME() ... 1194
NEXTKEY() ... 1194
NOSNOW() ... 1194
ORDBAGEXT() ... 1195
ORDBAGNAME() ... 1195
ORDCREATE() ... 1195
ORDDESTROY() ... 1195
ORDFOR() ... 1196
ORDKEY() ... 1196
ORDLISTADD() ... 1196
ORDLISTCLEAR() ... 1196
ORDLISTREBUILD() ... 1196
ORDNAME() ... 1197
ORDNUMBER() ... 1197
ORDSETFOCUS() ... 1197
OS() ... 1197
OUTERR() ... 1197
OUTSTD() ... 1197
PAD?() ... 1198
PCOL() ... 1198
PROW() ... 1198
QOUT() ... 1198
RAT() ... 1199
RDDLIST() ... 1199
RDDNAME() ... 1199
RDDSETDEFAULT() ... 1199
READINSERT() ... 1199
READMODAL() ... 1199
READVAR() ... 1200
RECNO() ... 1200
RECSIZE() ... 1200
REPLICATE() ... 1200
RESTSCREEN() ... 1200
RIGHT() ... 1201
RLOCK() ... 1201
ROUND() ... 1201
ROW() ... 1201
RTRIM() ... 1201
SAVESCREEN() ... 1201
SCROLL() ... 1202
SECONDS() ... 1202
SELECT() ... 1202
SET() ... 1202
SETBLINK() ... 1203
SETCANCEL() ... 1203

SETCOLOR() ... 1203
SETCURSOR() ... 1203
SETKEY() ... 1203
SETMODE() ... 1204
SETPOS() ... 1204
SETPRC() ... 1204
SOUNDEX() ... 1204
SPACE() ... 1204
SQRT() ... 1204
STR() ... 1205
STRTRAN() ... 1205
STUFF() ... 1205
SUBSTR() ... 1205
TIME() ... 1206
TONE() ... 1206
TRANSFORM() ... 1206
TYPE() ... 1206
UPDATED() ... 1207
UPPER() ... 1207
USED() ... 1207
VAL() ... 1207
VALTYPE() ... 1207
YEAR() ... 1207

nB functions ... 1208

ACCEPT() ... 1208
ACHOICE() ... 1208
ACHOICEWINDOW() ... 1208
ALERTBOX() ... 1209
ATB() ... 1209
BCOMPILE() ... 1210
BUTTON() ... 1210
COLORARRAY() ... 1210
COORDINATE() ... 1210
COPYFILE() ... 1211
DBAPP() ... 1211
DBCLOSE() ... 1211
DBCONTINUE() ... 1212
DBCOPY() ... 1212
DBCOPYSTRUCT() ... 1212
DBCOPYXSTRUCT() ... 1212
DBDELIM() ... 1212
DBISTATUS() ... 1213
DBISTRUCTURE() ... 1213
DBJOIN() ... 1213
DBLABELFORM() ... 1214
DBLIST() ... 1214
DBLOCATE() ... 1214
DBOLDCREATE() ... 1215
DBPACK() ... 1215
DBSDF() ... 1215
DBSORT() ... 1216
DBTOTAL() ... 1216
DBUPDATE() ... 1216
DBZAP() ... 1217
DISPBOXCOLOR() ... 1217
DISPBOXSHADOW() ... 1217
DIR() ... 1217
DOC() ... 1218

DOTLINE() 1218
DTEMONTH() 1218
DTEWEEK() 1218
EX() 1218
GET() 1219
GVADD() 1219
GVDEFAULT() 1219
GVFILEDIR() 1219
GVFILEEXIST() 1219
GVFILEEXTENTION() 1220
GVSUBST() 1220
HTF() 1220
ISFILE() 1220
ISWILD() 1220
ISMEMVAR() 1221
ISCONSOLEON() 1221
ISPRINTERON() 1221
KEYBOARD() 1221
LISTWINDOW() 1221
MEMOWINDOW() 1221
MEMPUBLIC() 1222
MEMRELEASE() 1222
MEMRESTORE() 1222
MEMSAVE() 1222
MENUPROMPT() 1222
MENUTO() 1222
MESSAGELINE() 1223
MOUSESCRSAVE() 1223
MOUSESCRRESTORE() 1223
PICCHRMAX() 1223
QUIT() 1223
READ() 1224
RF() 1224
RPT() 1224
RPTMANY() 1224
RPTTRANSLATE() 1225
RUN() 1225
SAY() 1225
SETCOLORSTANDARD() 1225
SETFUNCTION() 1225
SETMOUSE() 1225
SETOUTPUT() 1226
SETRPTEJECT() 1226
SETRPTLINES() 1227
SETVERB() 1227
SETVERB("EXACT") (obsolete) 1227
SETVERB("FIXED") 1228
SETVERB("DECIMALS") 1228
SETVERB("DATEFORMAT") 1228
SETVERB("EPOCH") 1228
SETVERB("PATH") 1228
SETVERB("DEFAULT") 1229
SETVERB("EXCLUSIVE") 1229
SETVERB("SOFTSEEK") 1229
SETVERB("UNIQUE") (obsolete) 1229
SETVERB("DELETED") 1229
SETVERB("CANCEL") 1229
SETVERB("TYPEAHEAD") 1229

SETVERB("COLOR") 1229
SETVERB("CURSOR") 1230
SETVERB("CONSOLE") 1230
SETVERB("ALTERNATE") 1230
SETVERB("ALTFILE") 1231
SETVERB("DEVICE") 1231
SETVERB("EXTRA") 1231
SETVERB("EXTRAFILE") 1231
SETVERB("PRINTER") 1231
SETVERB("PRINTFILE") 1231
SETVERB("MARGIN") 1231
SETVERB("BELL") 1231
SETVERB("CONFIRM") 1231
SETVERB("ESCAPE") 1232
SETVERB("INSERT") 1232
SETVERB("EXIT") 1232
SETVERB("INTENSITY") 1232
SETVERB("SCOREBOARD") 1232
SETVERB("DELIMITERS") 1232
SETVERB("DELIMCHARS") 1233
SETVERB("WRAP") 1233
SETVERB("MESSAGE") 1233
SETVERB("MCENTER") 1233
STRADDEXTENTION() 1233
STRCUTEXTENTION() 1233
STRDRIVE() 1233
STREXTENTION() 1234
STRFILE() 1234
STRFILEFIND() 1234
STRGETLEN() 1234
STRLISTASARRAY() 1234
STROCCURS() 1234
STRPARENT() 1234
STRPATH() 1235
STRTEMPPATH() 1235
STRXTOSTRING() 1235
TB() 1235
TEXT() 1236
TGLINSERT() 1236
TIMEX2N() 1236
TIMEN2H() 1236
TIMEN2M() 1236
TIMEN2S() 1237
TRUESETKEY() 1237
WAITFILEEVAL() 1237
WAITFOR() 1237
WAITPROGRESS() 1237
Normal command substitution 1237
nB command substitution functions 1247
RPT: the nB print function 1251
Memvars and fields 1251
Commands 1251
Examples 1253
How can I... 1254
The source files 1255

Dos xBase 1134
Composition 1137
How to use nB 1139
Status line 1140
The dot line 1140
The menu system 1140
The text editor DOC() 1149
The help text file 1150
Macro 1150
Data types 1154
Operators 1160
Delimiters 1161
Code blocks 1162
Standard functions 1162
nB functions 1208
Normal command substitution 1237
nB command substitution functions 1247
RPT: the nB print function 1251
How can I... 1254
The source files 1255

nB[1] (‘‘nano Base’’: ‘‘n’’ = ‘‘nano’’ = 10**(-9) = ‘‘very little’’) is a little Dos xBase written in Clipper 5.2 that can help to access ‘.DBF’ file created with different standards.

nB is:

- a dot command interpreter,
- a menu driven xBase,
- a xBase program interpreter.

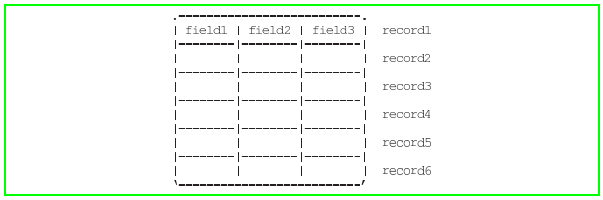
Dos xBase

This section is a brief description of the functionality of a typical Dos xBase.

The first purpose of a xBase program is to handle data inside a ‘.DBF’ file. These files may be indexed with the help of index files and more ‘.DBF’ files may be linked with a relation to obtain something like a relational database.

.DBF files

‘.DBF’ files are files organised in a table structure:



The lines of this table are records and the columns are fields. Records are numbered starting from the first that is number 1.

Columns are defined as fields and fields are distinguished by name and these names are saved inside the ‘.DBF’ file.

Every field (column) can contain only one specified kind of data with a specified dimension:

- ‘**C**’, character, originally the maximum dimension was 254 characters, minimum is 1;
- ‘**N**’, numeric, a numeric field that can contain also sign and decimal values;

- ‘**D**’, date, a field dedicated to date information;
- ‘**L**’, logic, a filed that may contain only ‘**T**’ for True or ‘**F**’ for False used as a boolean variable;
- ‘**M**’, memo, a character field with no predefined dimension, not allocated directly inside the ‘.DBF’, but inside a ‘.DBT’ file, automatically linked.

No other field type is available for a typical xBase ‘.DBF’ file.

To access the data contained inside a ‘.DBF’ file the following list of action may be followed:

- Open a ‘.DBF’ file inside the current area, where these areas are something like file handlers.
- After the ‘.DBF’ file is opened, it referenced only by the alias name that usually correspond to the original filename without extention.
- Move the record pointer to the desired location.
- Lock the current record to avoid access from other users.
- Do some editing with the data contained inside the current record using the field names like they were variables.
- Release the lock.
- Move the record pointer to another desired location.
- Lock the current record to avoid access from other users.
- ...
- Close the alias.

Before you go further, you have to understand that:

- A ‘.DBF’ file is opened using a free WORK AREA that may be associated to the concept of the file handler.
- The ‘.DBF’ file is opened with a alias name that permit to open the same ‘.DBF’ file more times when using different alias names.
- After the ‘.DBF’ file is opened, we don’t speak any more of file, but alias.
- If the work area "n" is used from the alias "myAlias", speaking of work area "n" or of alias "myAlias" is the same thing.

Index files

‘.DBF’ files are organised with record number, that is, you can reach a specific record and not a specific information unless that you scan record by record.

To obtain to "see" a ‘.DBF’ file somehow logically ordered (when physically it is not), index files are used.

A index file, also called INDEX BAG, is a file that contains one or more indexes

Indexes are rules by which a ‘.DBF’ file may be seen ordered.

A typical index file may contain only one index.

A index file may have the following extention:

- ‘.NDX’, single index, dBase III and dBase III plus;
- ‘.NTX’, single index, Clipper;
- ‘.MBX’, multiple index, dBase IV;
- ‘.CDX’, multiple index, FoxPro.

Every index file may be used only in association with the ‘.DBF’ for what it was made. The problem is that normally there is no way to avoid errors when the user try to associate the right ‘.DBF’ file with the wrong index.

To access the data contained inside a ‘.DBF’ file the following list of action may be followed:

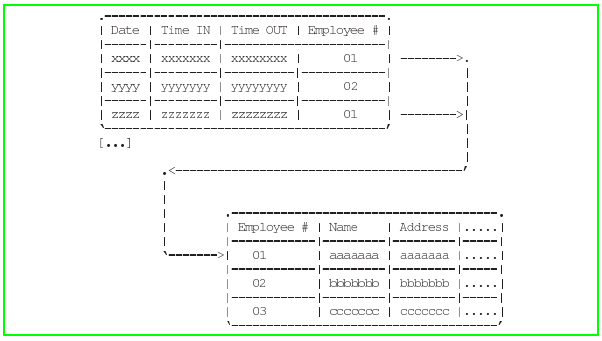
- Open a '.DBF' file.
- Open a index file.
- Select a particular order.
- Search for a key or move the record pointer on a different way.
- Lock the current record to avoid access from other users.
- Do some editing with the data contained inside the current record using the field names like they were variables.
- Release the lock.
- Move the record pointer to another desired location.
- Lock the current record to avoid access from other users.
- ...
- Close the alias.

Before you go further, you have to understand that:

- As orders are contained inside a INDEX BAG file physically distinguished form the '.DBF' file, it may happen that a '.DBF' file is wrongly opened and edited without the index. In this case, the INDEX BAG is not updated and when the INDEX BAG will be opened, the records contained inside the '.DBF' file may not correspond.
- For the same reason, an improper program termination may result in an incomplete data update. That is: '.DBF' file may be all right, INDEX BAG not.
- This is why xBase programs are "weak" relational databases or they are not relational databases at all.
- When troubles occurs, indexes must be rebuild.

Relations

Many '.DBF' files with indexes may be opened simultaneously. Data contained inside more '.DBF' files may be somehow connected together. See the example.



The first '.DBF' file contains some data that refers to an Employee number that may appear repeated on more records.

Employee informations are stored inside another '.DBF' file that contains only one record for every employee.

Establishing a relation from the first '.DBF' file to the second, moving the record pointer of the first '.DBF' file, that is the first alias, the record pointer of the second, the child alias, is moved automatically to the record containing the right data.

The relation is an expression that should result in a number if the child alias is opened without index, or in a valid index key if the child alias is opened with an index.

To relate two '.DBF' files the following list of action may be followed:

- Open the first '.DBF' file.
- Open a index file for the first alias.
- Select a particular order.

- Open the second '.DBF' file.
- Open a index file for the second alias.
- Select a particular order.
- Select the first alias.
- Define a relation form the first alias and the second alias: the child alias.
- Search for a key or move the record pointer of the first alias (don't care about the Child alias).
- Lock the current record to avoid access from other users.
- If data contained inside the Child alias should be edited (usually it doesn't happen), lock the current record of the Child alias.
- Do some editing with the data contained inside the current record using the field names like they were variables.
- Release the lock (also with the Child alias if a lock was made).
- Move the record pointer to another desired location.
- Lock the current record to avoid access from other users.
- [...]
- Release the relation.
- Close the Child alias.
- Close the first alias.

As may be seen, relations are not saved inside files, but are obtained with lines of code.

Composition

nB is composed from the following files, where ***xx*** is the the version code.

NBASExx1.ZIP	EXEs for small PCs
NBASExx2.ZIP	Runtime EXEs for small PCs
NBASExx3.ZIP	EXEs for i286 with 2M+
NBASExx4.ZIP	DOCs
NBASExx5.ZIP	EXAMPLEs
NBASExx6.ZIP	SRCs for version 96.06.16
NBASExx7.ZIP	SRCs for the current version

Every archive file contains:

'COPYING.TXT'	GNU General Public License version 2 in Dos text format.
'README.TXT'	the readme file.
'FILE_ID.DIZ'	definition.

The file 'NBASExx1.ZIP' contains also the following files.

'NB.EXE'	the executable program for DBFNTX and DBFNDX files, linked with RTLINK.
'NB.HLP'	this manual in "Help Text File" format.

The file NBASExx2.ZIP contains also the following files.

'NB.EXE'	the run-time to execute macro programs for DBFNTX and DBFNDX files handling, linked with RTLINK.

The file 'NBASExx3.ZIP' contains also the following files.

'NB.EXE'	the executable program for DBFCDX, DBFMDX, DBFNDX and DBFNTX files, linked with EXOSPACE.
'NB.HLP'	the user manual in "Help Text File" format.

The file 'NBASExx4.ZIP' contains also the following files.

'NB.PRN'	the user manual in printed text format.
'NB.RTF'	the user manual in RTF format.
'NB.TXT'	the user manual in ASCII text format.
'NB.HTM'	the user manual in HTML format.

The file 'NBASExx5.ZIP' contains also the following files.

'_ADDRESS.DBF'	an example database file.
'_ADDRESS.NTX'	index file associated to '_ADDRESS.DBF'.
'_ADDRESS.LBL'	a label form file used to print data contained inside '_ADDRESS.DBF' .
'_ADDRESS.FRM'	a report form file used to print data contained inside '_ADDRESS.DBF' .
'_ADDRESS.RPT'	a RPT text file used to print data contained inside '_ADDRESS.DBF' .
'_MAINMNU.&'	a macro program source example of a menu that executes some others macro programs. This example is made to demonstrate how nB can execute directly a source code without compiling it. This example is made only to taste it: it is very slow and only a speedy machine can give the idea of it.
'0MAINMNU.&'	a macro program source example of a menu that executes some others macro programs. It is the same as '_MAINMNU.&' but it is made to start the execution of the compiled macros.
'0MAINMNU.NB'	compiled macro program '0MAINMNU.&'
'0MAINMNU.BAT'	a batch file to show how to run the execution of '0MAINMNU.NB'
'1ADDRESS.&'	a macro program source example for handling a '.DBF' file containing addresses in various ways.
'1ADDRESS.NB'	compiled macro '1ADDRESS.&' .
'2ADDRESS.&'	a macro program source example for handling a '.DBF' file containing addresses in various ways: a little bit more complicated than 1ADDRESS.&.
'2ADDRESS.NB'	compiled macro '2ADDRESS.&' .
'3ADDRESS.&'	a macro program source example for handling a '.DBF' file containing addresses in various ways: a little bit more complicated than '2ADDRESS.&' .
'3ADDRESS.NB'	compiled macro '3ADDRESS.&'.
'4ADDRESS.&'	a macro program source example for handling a '.DBF' file containing addresses in various ways: a little bit more complicated than '3ADDRESS.&' .
'4ADDRESS.NB'	compiled macro '4ADDRESS.&' .
'ABIORITM.&'	a macro program source example for calculating the personal bio wave.
'ABIORITM.NB'	compiled macro 'ABIORITM.&' .
'_STUDENT.DBF'	a '.DBF' file used inside the BSTUDENT macro example.
'_STUDENT.NTX'	index file used for '_STUDENT.DBF' .
'_STUDSTD.DBF'	a '.DBF' file used inside the BSTUDENT macro example.
'_STUDENT.RPT'	a RPT text file used to print data contained inside '_STUDENT.DBF' .
'_STUDSTD.RPT'	a RPT text file used to print data contained inside '_STUDSTD.DBF' .
'BSTUDENT.&'	a macro program source example for students evaluation: a description about students is obtained linking other standard descriptions.
'BSTUDENT.NB'	compiled macro 'BSTUDENT.&' .
'CBATMAKE.&'	a macro program source example to generate a batch file to be used to back up an entire hard disk.
'CBATMAKE.NB'	compiled macro 'CBATMAKE.&' .
'BROWSE.&'	a macro program source example to start an automatic browse.
'BROWSE.NB'	compiled macro 'BROWSE.&' .
'BROWSE.BAT'	batch file to start a '.DBF' browse with the BROWSE macro program.
'MENU.&'	a macro program source example for a Dos menu.
'MENU.NB'	compiled macro 'MENU.&' .
'MENU.BAT'	batch file to use the MENU macro.

The file 'NBASExx6.ZIP' contains also the following files: source code for the version 96.06.16.

'NB.PRG'	the main source file for version 96.06.16.
'NB_REQ.PRG'	the source file containing links to all the standard functions.
'NB.LNK'	link file for compilation.
'NB_NRMAL.RMK'	rmake file to compile with RTLink.
'NB_EXOSP.RMK'	rmake file to compile with Exospace.
'NB_RUNTI.RMK'	rmake file to compile with RTLink defining RUNTIME to obtain a small nB runtime version.
'MACRO.LNK'	link file to compile and link a macro.
'MACRO.RMK'	rmake file to compile and link a macro.

The file 'NBASExx7.ZIP' contains also the following files: source code for the current version.

'NB.PRG'	the main source file.
'REQUEST.PRG'	the source file containing links to all the Clipper functions.
'STANDARD.PRG'	the source file for standard functions.
'EXTRA.PRG'	the source file for other standard functions.
'STANDARD.CH'	general include file that substitutes all include file normally used for normal Clipper compilations.
'NB.CH'	include file specific for nB.
'NB.LNK'	link file for compilation.
'NB_RUNTI.LNK'	link file for runtime compilation.
'NB_NRMAL.RMK'	rmake file to compile with RTLink.
'NB_EXOSP.RMK'	rmake file to compile with Exospace.
'NB_RUNTI.RMK'	rmake file to compile with RTLink defining RUNTIME to obtain a small nB runtime version.
'MACRO.CH'	include file to compile and link a macro.
'MACRO.LNK'	link file to compile and link a macro.
'MACRO.RMK'	rmake file to compile and link a macro.
'CLIPMOUSE.ZIP'	a simple free library for mouse support under Clipper (c) 1992 Martin Brousseau.

How to use nB

nB normal syntax is:

```
nB [nB_parameters] [macro_filename] [macro_parameters]
```

To run nB, just type the word "NB" and press [*Enter*] to execute. It will run in command mode, this means that it will look like an old xBASE command prompt.

To run the program as a macro interpreter, type the word NB followed from the macro file name with extention (no default extention is supposed). If parameters are given, after the macro file name, these will be available inside the public variables: c_Par1, c_Par2, ..., c_Par9. c_Par0 will contain the macro file name (see the macro file BROWSE.&). nB will terminate execution when the macro terminates.

These parameters are available for nB:

-c	Suppress the copyright notice. It is usefull when using nB for macro interpretation.
-w	Suppress the "Wait-Wheel" if not desired. It is the "Wheel" that appears at top-left when a macro is interpreted or other long elaborations are executed.
-?	Shows a short help.

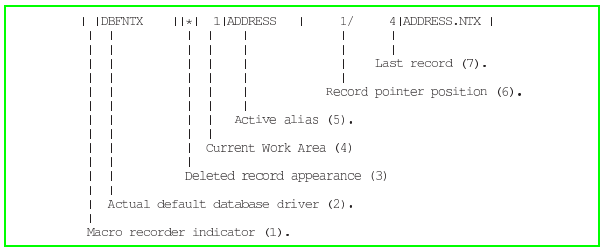
nB macro "compilation" syntax is:

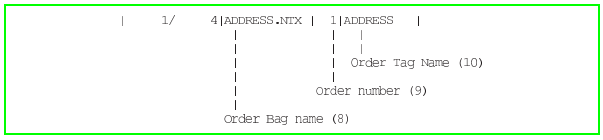
```
nB -m source_macro_filename [destination_macro_filename]
```

With the -m parameter, nB "compiles" the ASCII ***source_macro_filename*** into ***destination_macro_filename***.

Status line

nB shows a "status line" at the top of the screen when the nB command prompt or the menu system is active. It shows some important informations.





(1) This is the place for the macro recorder indicator. The symbol used is "&". Blank means that the macro recorder is OFF; & blinking means that the macro recorder is ON; & fixed means that the macro recorder is PAUSED.

(2) The name of the default database driver. It is not necessarily the database driver for the active alias; it is only the database driver that will be used for the next open/create operation.

(3) An asterisk (*) at this position indicates that SET DELETED is OFF. This means that deleted records are not filtered. When a BLANK is in this place, SET DELETED is ON, so that deleted records are filtered.

(4) The active work area number, that is, the area of the active alias.

(5) The active alias name. Note that the alias name is not necessarily equal to the '`.DBF`' file name.

(6) The actual record pointer position for the active alias.

(7) The number of records contained inside the active alias.

(8) The Order Bag name; that is the index file name.

(9) The order number.

(10) The order tag (name). When DBFNTX database driver is used, it correspond to the Order Bag name.

The dot line

Starting nB without parameters, the dot line appears. This is the place where commands in form of functions may be written and executed like a old xBase.

The functions written inside the command line that don't result in an error, are saved inside a history list. This history list may be recalled with [*F2*] and then the selected history line may be reused (eventually edited). Key [*up*]/[*down*] may be used to scroll inside the history list without showing the all list with [*F2*].

[*Enter*] is used to tell nB to execute the written function.

As the dot line is not an easy way to use such a program, a menu is available pressing [*F10*] or [*Alt M*]. The [*F10*] key starts the ASSIST() menu. This menu may be started also entering the name of the function: "ASSIST()".

nB includes a simple built-in text editor: DOC(). It may be started from the dot line entering "DOT()". No special key is dedicated to start this function.

The menu system

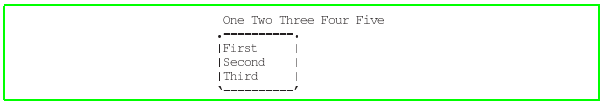
The nB menu system appears differently depending on the place where it is "called". When available, the menu system appears pressing [*Alt M*] or [*F10*].

The Menu system is organised into horizontal menu, vertical menu, and pop-up menu.

The horizontal menu contains selectable items organised horizontally:

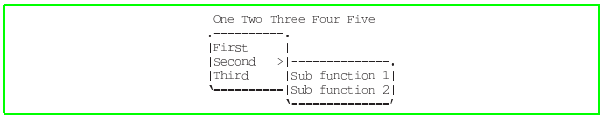
The cursor may be moved on a different position using arrow keys [*Left*]/[*Right*]; [*Esc*] terminates the menu; [*Enter*] opens a vertical menu.

The vertical menu contains selectable items organised vertically:



The cursor may be moved on a different position using arrow keys [*Up*]/[*Down*]; the arrow keys [*Left*]/[*Right*] change the vertical menu; [*Esc*] closes the vertical the menu; [*Enter*] starts the selected menu function.

The vertical menu contains selectable items organised vertically:



The cursor may be moved on a different position using arrow keys [*Up*]/[*Down*]; [*Esc*] closes the pop-up the menu; [*Enter*] starts the selected menu function.

The following sections describe the menu system.

Menu File

The menu File contains important function on '`.DBF`' file, indexes, relations and Replaceable database drivers.

For database files are considered two aspects: the physical aspect, and the logical alias. When a '`.DBF`' file is opened, it becomes a alias.

Indexes are considered as index files and index orders.

It follows a brief menu function description.

Change directory

Changes the actual drive and directory.

File .DBF

Contains a pop-up menu for '`.DBF`' operations.

New .DBF

A '`.DBF`' file is a table where columns, called Fields, must be specified and lines, called records, are added, edited and deleted by the program.

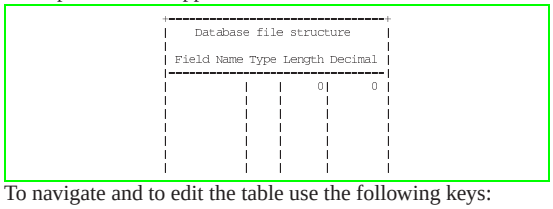
Field characteristics are:

NAME	the field name must be unique inside the same file, it is composed of letters, number and underscore (_), but it must start with a letter and it is not case sensitive.
TYPE	the field type determinates the type of data it can hold.
LENGTH	is the field total length in characters; it doesn't matter of the type of data.
DECIMAL	is the length of positions after decimal point. This information is used normally for numeric fields. In this case, take note that the DECIMAL length, together with the decimal point, will subtract space for the integer part of the number from the total LENGTH of the filed.

Field Types:

C Character	it is a text field long LENGTH characters.
N Numeric	it is a numeric field long LENGTH characters with DECIMAL characters for decimal positions. Note that if LENGTH is 4 and DECIMAL is 0 (zero), the field may contain integers from -999 to 9999; but if LENGTH is 4 and DECIMAL 1, the field may contain numbers from -9.9 to 99.9: two position for the integer part, one position for the decimal point and one position for decimal.
D Date	it is a date field: it contains only dates; the length should not be specified as it is automatically 8.
L Logic	it is a logical (boolean) field: it contains only TRUE, represented by "Y" or "T", or FALSE, represented by "N" or "F". The length should not be specified as it is automatically 1.
M Memo	it is a character field with unknown dimension. It is recorded into a parallel file with '.DBT' extention. The original '.DBF' file holds a space for a pointer inside the '.DBT' file. The length of a Memo field is automatically 10 and is referred to the memo pointer.

After the function "NEW .DBF" is selected, a table for the field specifications appears.



To navigate and to edit the table use the following keys:

[*Up*]/[*Down*]/[*Left*][*Right*]	move the cursor one position (up, down, left or right);
[*PgUp*]	move to previous screen page;
[*PgDn*]	move to next screen page;
[*Ctrl PgUp*]	move to top of table;
[*Ctrl PgDn*]	move to bottom of table;
[*Ctrl Home*]	move to first column;
[*Ctrl End*]	move to last column;
[*Ctrl Enter*]	append a new empty line;
[*Ctrl F1*]	delete (cut) the current line and save a copy into the "clipboard";
[*Ctrl F2*]	copy current line into the table "clipboard";
[*Ctrl F3*]	insert (paste) the content of the "clipboard" in the current position;
[*Enter*]	start editing in the current position;
[*Esc*]	terminate;
[*x*]	any other key will be written in the current position.

When the editing is terminated, press [*Esc*] and a dialog box will ask for the file name and the RDD.

xBase files (.DBF) are not all equal, this way, when a new '.DBF' file si created, the RDD (Replaceable Database Driver) is asked. The normal RDD is DBFNTX, the one used by Clipper.

Modify .DBF structure

The modification of a '.DBF' file structure is a delicate matter if it contains data.

In fact, it is a data transfer from a source '.DBF' file to a destination '.DBF' file with a different structure. This way, the destination '.DBF' will be updated only for the fields with the same name of the source one. The position may be different, but names cannot be changed (not so easily).

Mistakes may be dangerous, so, before doing it, it is recommended a backup copy of the original '.DBF' file.

Open .DBF

When a '.DBF' file is opened, it becomes a alias, a logical file, placed inside a work area. The same '.DBF' file may be opened inside different areas with different alias names.

The required information to open the file are:

FILENAME	the physical file name.
ALIAS	the alias name. If not assigned, it becomes automatically the same of FILENAME without extention.
RDD	the Replaceable Database Driver to use to access to this file.
SHARED	a logical value: TRUE means that the file will be accessible to other users, FALSE means use exclusive.
READ ONLY	a logical value: TRUE means that the file will be only readable and no modification will be allowed, FALSE means that no restriction on editing will be made.

File .NTX

Contains a pop-up menu for physical indexes operations.

New .NTX / new tag

If the active area is used we have an active alias. In this case a index may be created. The index is a way to see the active alias ordered without changing the physical position of records.

There are two words to understand: ORDER and INDEXBAG. The index bag is the file that contains the information on the record ordering, the order is the rule followed to order the records. A index bag may contains one or more orders depending on the Replaceable Database Driver in use.

Typical '.NTX' file are index bag containing only one order.

Depending on the RDD in use the following field may be filled.

INDEX FILENAME	this is the name of the index bag.
KEY EXPRESSION	the expression that defines the rule for the record ordering.
ORDER NAME	this is the name to give to the order (tag) when the RDD permits to have a index bag containing more than one order. In the other case, the index bag name correspond to the order name.
FOR EXPRESSION	a FOR condition to filter records before indexing.

Open index

If a index file already exists, it can be associated to the active alias simply opening it.

Take note that the system is not able to verify if the index belong the active alias and if it is not so a error will result.

INDEX NAME	is the name of the index bag file to open.

Alias

Contains a pop-up menu for logical databases (alias) operations.

Select

Only one may be the active alias and with this function the active alias may be changed choosing from the list of used areas.

Selecting the area number zero, no alias is active.

Display structure

With this function the active alias structure may be viewed.

Close active alias

Selecting this function the active alias is closed. That is: the '.DBF' file and eventual indexes are closed.

Close all aliases

With this function all Aliases are closed.

Order

Contains a pop-up menu for logical indexes (orders).

Order list rebuild

This function rebuild the indexes opened and associated to the active alias.

Order set focus

This function permits to change the active order selecting form the ones opened and associated to the active alias.

Order list clear

This function closes all orders associated to the active alias.

Relation

Contains a pop-up menu for relations (links with other Aliases).

Set relation

This function permits to establish a relation between a alias and a Child alias showing as a result a unique database.

CHILD	is the alias name to connect to the active alias.
EXPRESSION	is the relation expression that specify the rule for the relation. The value of this expression is the key to access the Child alias: if this Child alias is accessed without index, it must be the record number, if this Child alias is accessed via index, it must be a valid index key.

Clear relation

This function eliminates any relation that originate form the active alias.

RDD default

Contains a pop-up menu for Replaceable Database Driver defaults.

Show actual RDD default

It simply shows the actual Replaceable Database Driver.

Set default RDD

Select a new default Replaceable Database Driver.

Menu Edit

The menu Edit contains functions to access data from the active alias (the actual area).

View

This function permits you to view the active alias with eventual relations as a table.

No edit is allowed.

To navigate the table use the following keys.

[*Enter*]	start field editing.
[*PgUp*]	show previous screen page.
[*PgDn*]	show next screen page.
[*Ctrl PgUp*]	show top of alias.
[*Ctrl PgDn*]	show bottom of file.
[*Ctrl Home*]	show the first column.
[*Ctrl End*]	show last column.

Edit/browse

This function permits you to edit the active alias with eventual relations as a table.

To navigate and edit the table use the following keys.

[*Enter*]	start field editing.
[*PgUp*]	show previous screen page.
[*PgDn*]	show next screen page.
[*Ctrl PgUp*]	show top of alias.
[*Ctrl PgDn*]	show bottom of file.
[*Ctrl Home*]	show the first column.
[*Ctrl End*]	show last column.
[*Ctrl Enter*]	append a new empty record.
[*Ctrl F2*]	copy the current record.
[*Ctrl F3*]	append and paste a record.
[*Ctrl F4*]	paste a previously copied record, overwriting the content of the current one.
[*Ctrl Y*]	delete or recall the current record.
[*Ctrl Del*]	delete or recall the current record.

When a memo field is edited:

[*Esc*]	cancel and close the memo window.
[*Ctrl Y*]	line delete.
[*Ctrl W*]	save and close the memo window.

Replace

The content of a Field of the active alias may be replaced with an expression.

The required data is:

FIELD TO REPLACE	the Field name to be replaced.
NEW VALUE EXPRESSION	the expression that obtain the new value for the selected Field.
WHILE EXPRESSION	the WHILE condition expression: the replacement continue until this expression results True. The constant '`.T.`' is ever True and is the default.
FOR EXPRESSION	the FOR condition expression: the replacement is made for all records that satisfy the condition. The constant '`.T.`' is ever True and is the default.

Recall

The records signed for deletion (deleted but still there), may be recalled (undeleted).

The required data is:

WHILE EXPRESSION	the WHILE condition expression: the record recall continue until this expression results True. The constant '`.T.`' is ever True and is the default.
FOR EXPRESSION	the FOR condition expression: the record recall is made for all records that satisfy the condition. The constant '`.T.`' is ever True and is the default.

Delete

Deletes (sign for deletion) a group of record depending on the required conditions.

The required data is:

WHILE EXPRESSION	the WHILE condition expression: the record deletion continue until this expression results True. The constant '`.T.`' is ever True and is the default.
FOR EXPRESSION	the FOR condition expression: the record deletion is made for all records that satisfy the condition. The constant '`.T.`' is ever True and is the default.

Pack

This function eliminates definitely records previously deleted (signed for deletion).

It may work only if the active alias was opened in exclusive mode.

Menu Report

The menu Report contains functions for data report (print). In particular, label files '`.LBL`' and report file '`.RPT`' may be created and used for printing. There is also another way to print, with the RPT() system that is available inside the nB internal editor DOC().

DBGOTOP()

Moves the record pointer for the active alias at the first logical record.

New label

With this function can be created a standard label file (.LBL under the dBaseIII standard).

Labels may be printed in more than one column and can contain 16 lines maximum.

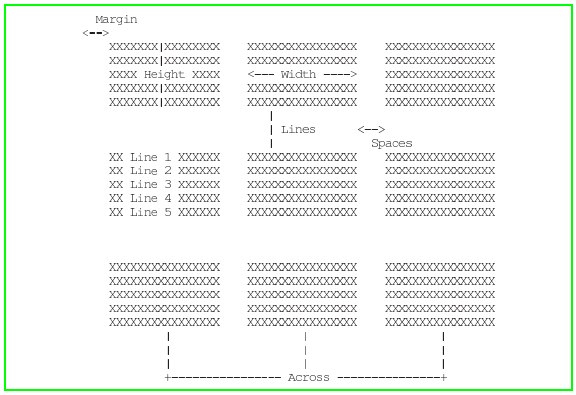
The label data is the following.

REMARK	a label remark that will not be printed.
HEIGHT	the label vertical dimension.
WIDTH	the label horizontal dimension.
MARGIN	the left margin in characters.
LINES	the vertical spacing between labels.
SPACES	the horizontal spacing between labels in characters.
ACROSS	the number of label columns.
LINE 1	The first line inside labels.
LINE n	The n-th line inside labels.
LINE 16	The 16th line inside labels.

The number of lines inside the labels depend on the HEIGHT and the maximum value is 16.

The label lines can contain constant string and/or string expressions.

See the example below.



Modify label

This function permits you to modify a label file.

Label form

This function permits you to print labels with the data provided by the active alias: one label each record.

The following data is required.

LABEL FILENAME	the label filename.
WHILE	the WHILE condition: the label printing goes on as long as this condition remain True.
FOR	the FOR condition: only the records from the active alias that satisfy the condition are used for the label print.

New report

This function permits you to create a standard report form file (.FRM under the dBaseIII standard).

The informations to create a '`.FRM`' file are divided into two parts: the head and groups; the columns.

The first part: head and groups, requires the folliwing informations:

PAGE WIDTH	the page width in characters.
LINES PER PAGE	the usable lines per page.
LEFT MARGIN	the left margin in characters.
DOUBLE SPACED?	double spaced print, yes or no.
PAGE EJECT BEFORE PRINT?	form feed before print, yes or no.
PAGE EJECT AFTER PRINT?	form feed after print, yes or no.
PLAIN PAGE?	plain page, yes or no.
PAGE HEADER	the page header, max 4 lines (the separation between one line and the other is obtained writing a semicolon, ";").
GROUP HEADER	the group title.
GROUP EXPRESSION	the group expression (when it changes, the group changes)
SUMMARY REPORT ONLY?	only totals and no columns, yes or no.
PAGE EJECT AFTER GROUP?	form feed when the group changes, yes or no.
SUB GROUP HEADER	sub group title.
SUB GROUP EXPRESSION	the sub group expression.

The second part: columns, requires the following informations structured in table form:

COLUMN HEADER	column head description (it can contain 4 lines separated with a semicolon).
CONTENT	the column expression.
WIDTH	the column width.
DEC.	the decimal length for numeric columns.
TOTALS	totals to be calculated, yes or no (usefull only for numeric columns).

To navigate and to edit the table use the following keys:

[*Up*]/[*Down*]/[*Left*][*Right*]	move the cursor one position (up, down, left or right);
[*PgUp*]	move to previous screen page;
[*PgDn*]	move to next screen page;
[*Ctrl PgUp*]	move to top of table;
[*Ctrl PgDn*]	move to bottom of table;
[*Ctrl Home*]	move to first column;
[*Ctrl End*]	move to last column;
[*Ctrl Enter*]	append a new empty line;
[*Ctrl F1*]	delete (cut) the current line and save a copy into the "clipboard";
[*Ctrl F2*]	copy current line into the table "clipboard";
[*Ctrl F3*]	insert (paste) the content of the "clipboard" in the current position;
[*Enter*]	start editing in the current position;
[*Esc*]	terminate;
[***x***]	any other key will be written in the current position.

When the editing is terminated, press [*Esc*] and a dialog box will ask for the name to give to the report form file.

Modify report

This function permits you to modify a standard report form file (.FRM under the dBaseIII standard).

Report form

This function permits you to print a report form with the data provided by the active alias.

The following data is required.

REPORT FORM FILENAME	the label filename.
WHILE	the WHILE condition: the form printing goes on as long as this condition remain True.
FOR	the FOR condition: only the records from the active alias that satisfy the condition are used for the report form print.

Create/modify/print text

This function activates the text editor.

Menu HTF

The menu Htf helps on creating and accessing the "Help Text Files". This name, help text file, is just the name given to it.

A text (Ascii) file prepared like this manual may be transformed into a "Help Text File" that is a simple text with pointers.

Open help text file

This function permits to open a Help Text File and browse it. The Help Text File name is required.

New help text file

This function permits to create a new "Help Text File" that is a help file under the nB style.

The source is an Ascii file where three kind of information are available: Normal text, Indexes and pointers.

Indexes and Pointers are word or phrases delimited with user defined delimiters; indexes are placed inside the text to indicate an argument, pointers are placed inside the text to indicate a reference to indexes.

Inside this manual, indexes are delimited with ## and ##, so the titles are here indexes; pointers are delimited with < and >.

Only one index per line is allowed, only one pointer per line is allowed.

The Delimiters used do identify indexes and pointers are user defined; the _start_ identifier symbol can be equal to the _end_ identifier symbol. The symbols used for indexes cannot be used for the pointers.

So, the informations required are:

SOURCE TEXT FILENAME	the filename of the text source file.
DESTINATION FILENAME	the filename of the destination Help Text File (suggested '.HLP' extention).
INDEX START CODE	the index start symbol; suggested ##.
INDEX END CODE	the index end symbol; suggested ##.
POINTER START CODE	the pointer start symbol; suggested <.
POINTER END CODE	the pointer end symbol; suggested >.

New HTML file

This function permits to create a new HTML file form a text file formatted to obtain a HTF file.

The informations required are:

SOURCE TEXT FILENAME	the filename of the text source file.
DESTINATION FILENAME	the filename of the destination Help Text File (suggested '.HLP' extention).
INDEX START CODE	the index start symbol; suggested ##.
INDEX END CODE	the index end symbol; suggested ##.
POINTER START CODE	the pointer start symbol; suggested <.
POINTER END CODE	the pointer end symbol; suggested >.
HTML TITLE	the title for the html page.

Menu Macro

The menu Macro helps on creating macros (programs) with a macro recorder, a macro "compiler" and a macro executor.

Start recording

This function simply starts or pause the macro recording. The menu items that end with "&", may be recorded by this macro recorder.

Save recording

A recorded macro may be saved into a ASCII file that may be later modified or simply used as it is. The filename is requested.

Erase recording

While recording or when the macro recorder is paused, it is possible to erase all previous recording with this function.

Edit recording

While recording or when the macro recorder is paused, it is possible to edit all previous recording, for example adding more comments or simply to see what the recorder does.

Macro compilation

A macro file (a program) contained inside a ASCII file, may be compiled into a different file format to speed up execution. The source filename and the destination filename are requested.

Load + execute macro

A macro file (a program) in ASCII form or compiled, may be executed.

A macro file may require some parameters.

This function asks for the macro filename to start and the possible parameter to pass to it.

Menu Info

The menu Info is the information menu.

ABOUT	a brief copyright notice.
MANUAL BROWSE	starts the browse of 'NB.HLP', the nB Help Text File manual if it is present in the current directory or it is found in the PATH (the Dos SET PATH).
[F1] HELP	[F1] reminder.
[F3] ALIAS INFO	[F3] reminder. It shows all the available information on the active alias.
[F5] SET OUTPUT TO	[F5] reminder. It defines the output peripheral or file.

Menu Doc

This menu actually appears only inside the DOC() function, the nB text editor.

New

It starts the editing of a new empty text.

Open

It opens for editing a new textfile.

Save

It saves the text file under editing.

Save as

It saves the text file under editing asking for a new name.

Set output to

It permits to change the default output peripheral: the default is the screen.

Print as it is

It prints on the output peripheral the content of the text as it is.

Print with RPT() once

It prints on the output peripheral the content of the text only once replacing possible text variables.

Print with RPT() std

It prints on the output peripheral the content of the text repeating this print for every record contained inside the archive alias.

Exit DOC()

Terminates the use of DOC() the text/document editing/print function.

The text editor DOC()

The function Doc() activates a simple text editor usefull to build some simple reports.

Inside this function a menu is available and is activated pressing [*Alt M*] or [*F10*]. The Doc() menu is part of the nB menu system.

DOC() may handle text files of a teorical maximum of 64K.

DOC() may be particularly useful to create formatted text with variables identified by CHR(174) and CHR(175) delimiters: when an active alias exists, [*F2*] gives a list of insertable fields.

[*Esc*]	Exit DOC().
[*F1*]	Call the help.
[*F2*]	Field list.
[*up*] / [*Ctrl E*]	Line up.
[*down*] / [*Ctrl X*]	Line down.
[*left*] / [*Ctrl S*]	Character left.
[*right*] / [*Ctrl D*]	Character right.
[*Ctrl right*] / [*Ctrl A*]	Word left.
[*Ctrl left*] / [*Ctrl F*]	Word right.
[*Home*]	Line start.
[*End*]	Line end.
[*Ctrl Home*]	Top window.
[*Ctrl End*]	Bottom window.
[*PgUp*]	Previous window.
[*PgDn*]	Next window.
[*Ctrl PgUp*]	Document start.
[*Ctrl PgDn*]	End document.
[*Del*]	Delete character (right).
[*Backspace*]	Delete character Left.
[*Tab*]	Insert tab.
[*Ins*]	Toggle insert/overwrite.
[*Enter*]	Next line.
[*Ctrl Y*]	Delete line.
[*Ctrl T*]	Delete word right.
[*F10*] / [*Alt M*]	DOC() menu.

The help text file

nB provides a basic hypertext system to build simple help files. A source text file with "indexes" and "pointers" to indexes is translated into a "help text file" (a '`.DBF`' file); then, this file is browsed by nB.

The source file can have a maximum line width of 80 characters; each line can terminate with CR or CR+LF.

"Indexes" are string delimited by index delimiters (default "##"); "pointers" are string delimited by pointer delimiters (default "<" and ">") and refers to indexes.

Inside a text, indexes must be unique; pointers can be repeated anywhere. A text can contain a maximum of 4000 indexes.

Inside this manual, titles are delimited with "##" as they are indexes; strings delimited with "<" and ">" identify a reference to a title with the same string.

To browse a previously created Help Text File, use the following keys:

[*Esc*]	Exit.
[*UpArrow*]	Move cursor up.
[*DownArrow*]	Move cursor down.
[*PgUp*]	Move cursor PageUp.
[*PgDn*]	Move cursor Pagedown.
[*Ctrl PgUp*]	Move cursor Top.
[*Ctrl PgDn*]	Move cursor Bottom.
[*Enter*]	Select a reference (pointer).
[< -]	Go to previous selected reference (pointer).
[- >]	Go to next selected reference (pointer).
[*Shift F3*]	Search for a new pattern.
[*F3*]	Repeat previous search.

Macro

nB can execute (run) macro files. There may be three kind of macro files: ASCII (usually with .& extention); "compiled" (usually with .NB extention); EXE files (compiled with Clipper and linked).

"Compiled" macro files are executed faster then the ASCII source files.

EXE macro files are the fastest.

Macro statements

The statements recognised from nB are very similar to Clipper, with some restrictions.

Note that: the FOR statement is not included; there is no function declaration; procedure calls cannot transfer variables; only public variables are allowed.

PROCEDURE

Procedures are the basic building blocks of a nB macro.

Procedures are visible only inside the current macro file.

The procedure structure is as follows:

```
PROCEDURE procedure_name
      statements...
      [RETURN]
      statements...
ENDPROCEDURE
```

A procedure definition begins with a PROCEDURE declaration followed with the ***procedure_name*** and ends with ENDPROCEDURE.

Inside the PROCEDURE - ENDPROCEDURE declaration are placed the executable ***statements*** which are executed when the procedure is called.

Inside the PROCEDURE - ENDPROCEDURE declaration, the RETURN statement may appear. In this case, encountering this RETURN statement, the procedure execution is immediately terminated and control is passed to the statement following the calling one.

The procedure definition do not permit to receive parameters from the calling statement.

DO PROCEDURE

There is only one way to call a procedure:

```
DO PROCEDURE procedure_name
```

When the statement DO PROCEDURE is encountered, the control is passed to the begin of the called PROCEDURE. After the PROCEDURE execution, the control is returned to the statement following DO PROCEDURE.

The procedure call do not permit to send parameters to the procedure.

BEGIN SEQUENCE

The BEGIN SEQUENCE - END structure permits to define a sequence of operation that may be broken.

Inside nB, this control structure is useful only because there is the possibility to break the execution and pass control over the end of it.

This way, encountering BREAK means: "go to end".

```
BEGIN SEQUENCE
      statements...
      [BREAK]
      statements...
END
```

Inside nB, error exception handling is not supported.

DO CASE

This is a control structure where only the statements following a True CASE condition are executed.

When the DO CASE statement is encountered, the following CASE statements are tested. The first time that a condition returns True, the CASE's statements are executed and then control is passed over the END case.

That is: only one CASE is taken into consideration.

If no condition is True, the statements following OTHERWISE are executed.

```
DO CASE
CASE lCondition1
      statements...
[CASE lCondition2]
      statements...
[OTHERWISE]
      statements...
END
```

WHILE

The structure WHILE - END defines a loop based on a condition: the loop is repeated until the condition is True.

The loop execution may be broken with the EXIT statement: it transfer control after the END while.

The LOOP statement may be use to repeat the loop: it transfer the control to the beginning of the loop.

```
WHILE lCondition
      statements...
      [EXIT]
      statements...
      [LOOP]
      statements...
END
```

IF

The IF - END control structure executes a section of code if a specified condition is True. The structure can also specify alternative code to execute if the condition is False.

```
IF lCondition1
      statements...
[ELSE]
      statements...
END
```

Variable declaration

Inside nB, variables are created using a specific function:

```
MEMPUBLIC( "cVarName" )
```

For example,

```
MEMPUBLIC( "Name" )
```

creates the variable Name.

The scope of the created variable is global and there is no way to restrict the visibility of it.

When a variable is no more needed or desired, it can be released:

```
MEMRELEASE( "cVarName" )
```

The variable declaration do not defines the variable type. Every variable may receive any kind of data; that is that the type depends on the type of data contained.

Macro structure

A nB macro must be organised as follow. There may be two situations: Macros with procedures and macros without procedures.

Macro with procedures:

```
PROCEDURE procedure_name_1
      statements...
      [RETURN]
      statements...
ENDPROCEDURE
PROCEDURE procedure_name_2
      statements...
      [RETURN]
      statements...
ENDPROCEDURE
...
...
DO PROCEDURE procedure_name_n
```

Macro without procedures:

```
statements...
statements...
statements...
statements...
statements...
```

nB Macros may be compiled with Clipper. To do so, the first structure example must be changed as follows:

```
#INCLUDE MACRO.CH

DO PROCEDURE procedure_name_nth
...
PROCEDURE procedure_name_1
      statements...
      [RETURN]
      statements...
ENDPROCEDURE
PROCEDURE procedure_name_2
      statements...
      [RETURN]
      statements...
ENDPROCEDURE
...
...
```

To compile a macro with Clipper, the macro file name can be changed into 'MACRO.PRG' and

```
RTLINK MACRO.RMK [Enter]
```

should be started.

Macro comments

A nB Macro source file can contain comments. only the "//" comment is recognised! This way: * and /*...*/ will generate errors!

ATTENTION: to simplify the macro interpretation, lines such as this:

```
qqout( "You can't do that // you can't do that!" )
```

will generate an error as the interpreter will read only:

```
qqout( "You can't do that
```

Sorry!

Macro long lines split

Inside a nB macro, long lines may be splitted using ";" (semicolon). Please note that: lines can only be splitted and not joined; a resulting command line cannot be longer then 254 characters.

The macro recorder

Inside the functions ASSIST() and DOC() is available the Macro recorder menu.

When a macro recording is started, a "&" appears on the left side of the status bar. It it blinks, the recording is active, if it is stable, the recording is paused.

The macro recording is not exactly a step-by-step recording of all action taken, but a translation (as good as possible) of what you have done.

The macro recorder is able to record only the menu functions that terminates with the "&" symbol and all what is inserted at the dot command line.

The macro recording can be viewed and edited during the recording. The macro recording can be saved into a text file (a macro file).

Data types

The data types supported in the nB macro language are the same as Clipper:

Array

Character

Code Block

Numeric

Date

Logical

Memo

NIL

Character

The character data type identifies character strings of a fixed length. The character set corresponds to: CHR(32) through CHR(255) and the null character, CHR(0).

Valid character strings consist of zero or more characters with a theoretical maximum of 65535 characters. The real maximum dimension depends on the available memory.

Character string constants are formed by enclosing a valid string of characters within a designed pair of delimiters. There are three possible delimiter pairs:

two single quotes like '***'string_constant'***';

two double quotes like '***"string_constant"***';

left and right square brackets like '***[string_constant]***'.

These three different kind of delimiters are available to resolve some possible problems:

I don't want it -> "I don't want it"

She said, "I love hin" -> 'She said, "I love hin"'

He said, "I don't want it" -> [He said, "I don't want it"]

The following table shows all operations available inside nB for character data types. These operations act on one or more character expressions and the result is not necessarily a character data type.

+	Concatenate.
-	Concatenate without intervening spaces.
==	Compare for exact equity.
!=, <>, #	Compare for inequity.
<	Compare for sorts before
<=	Compare for sorts before or same as.
>	Compare for sorts after.
>=	Compare for sorts after or same as.
:=	In line assign.
$	Test for substring existence.
ALLTRIM()	Remove leading and trailing spaces.
ASC()	Convert to numeric ASCII code equivalent.
AT()	Locate substring position.
CTOD()	Convert to date.
DESCEND()	Convert to complemented form.
EMPTY()	Test for null or blank string.
ISALPHA()	Test for initial letter.
ISDIGIT()	Test for initial digit.
ISLOWER()	Test for initial lowercase letter.
ISUPPER()	Test for initial uppercase letter.

LEFT()	Extract substring form the left.
LEN()	Compute string length in characters.
LOWER()	Convert letters to lowercase.
LTRIM()	Remove leading spaces.
PADC()	Pad with leading and trailing spaces.
PADL()	Pad with leading spaces.
PADR()	Pad with trailing spaces.
RAT()	Locate substring position starting from the right.
RIGHT()	Extract substring form the right.
RTRIM()	Remove trailing spaces.
SOUNDEX()	Convert to soundex equivalent.
SPACE()	Create a blank string of a defined length.
STRTRAN()	Search and replace substring.
STUFF()	Replace substring.
SUBSTR()	Extract substring.
TRANSFORM()	Convert to formatted string.
UPPER()	Convert letters to uppercase
VAL()	Convert to numeric.
VALTYPE()	Evaluates data type directly.

Memo

The memo data type is used to represent variable length character data that can only exist in the form of a database field.

Memo fields are not stored inside the main database file (.DBF) but inside a separate file (.DBT).

A memo field can contain up to 65535 characters, that is the same maximum dimension of character fields. In fact, originally xBases, couldn't have character string longer than 254 characters.

As here memo fields are very similar to long character strings, you may forget that there is a difference.

All the operations that may be applied to character strings, may be used with memo fields; the following functions may be use especially for memo fields or long character strings.

HARDCR()	Replace soft with hard carriage returns.
MEMOEDIT()	Edit contents.
MEMOLINE()	Extract a line of a text.
MEMOREAD()	Read form a disk text file.
MEMOTRAN()	Replace soft and hard carriage returns.
MEMOWRIT()	Write to disk text file.
MLCOUNT()	Count lines.
MLPOS()	Compute position.

Date

The date data type is used to represent calendar dates.

Supported dates are from 0100.01.01 to 2999.12.31 and null or blank date.

The appearance of a date is controlled from SETVERB("DATEFORMAT"). The default is "dd/mm/yyyy" and it may easily changed for example with SETVERB("DATEFORMAT", "MM/DD/YYYY") to the US standard.

There is no way to represent date constants; these must be replaced with the CTOD() function. For example if the date 11/11/1995 is to be written, the right way is:

```
CTOD( "11/11/1995" )
```

The character string "11/11/1995" must respect the date format defined as before explained.

The function CTOD() will accept only valid dates, and null dates:

```
CTOD( "" )
```

A null date is ever less than any other valid date.

The following table shows all operations available inside nB for date data types. These operations act on one or more date expressions and the result is not necessarily a character data type.

+	Add a number of days to a date.
-	Subtract days to a date.
==	Compare for equity.
!=, <>, #	Compare for inequity.
<	Compare for earlier
<=	Compare for earlier or same as.
>	Compare for later.
>=	Compare for later or same as.
:=	In line assign.
CDOW()	Compute day of week name.
CMONTH()	Compute month name.
DAY()	Extract day number.
DESCEND()	Convert to complemented form.
DOW()	Compute day of week.
DTOC()	Convert to character string with the format defined with SETVERB("DATEFORMAT").
DOTOS()	Convert to character string in sorting format (YYYYMMDD).
EMPTY()	Test for null date.
MONTH()	Extract month number.
VALTYPE()	Evaluates data type directly.
YEAR()	Extract entire year number, including century.

Numeric

The numeric data type identifies real number. The theoretical range is form 10^-308 to 10^308 but the numeric precision is guaranteed up to 16 significant digits, and formatting a numeric value for display is guaranteed up to a length of 32 (30 digits, a sign, and a decimal point). That is: numbers longer than 32 bytes may be displayed as asterisks, and digits other then most 16 significant ones are displayed as zeroes.

Numeric constants are written without delimiters. The following are valid constant numbers:

12345

12345.678

-156

+1256.789

-.789

If a numeric constant is delimited like character strings, it becomes a character string.

The following table shows all operations available inside nB for numeric data types. These operations act on one or more numeric expressions and the result is not necessarily a numeric data type.

+	Add or Unary Positive.
-	Subtract or Unary Negative.
*	Multiply.
/	Divide.
%	Modulus.
^, **	Exponentiate.
==	Compare for equity.
!=, <>, #	Compare for inequity.
<	Compare for less than.
>=	Compare for less than or equal.
>	Compare for greater than.
>=	Compare for greater than or equal.
:=	In line assign.
ABS()	Compute absolute value.
CHR()	Convert to ASCII character equivalent.
DESCEND()	Convert to complemented form.
EMPTY()	Test for zero.
EXP()	Exponentiate with e as the base.
INT()	Convert to integer.
LOG()	Compute natural logarithm.
MAX()	Compute maximum.
MIN()	Compute minimum.

ROUND()	Round up or down()
SQRT()	Compute square root.
STR()	Convert to character.
TRANSFORM()	Convert to formatted string.
VALTYPE()	Evaluates data type directly.

Number appearence may be affected by SETVERB("FIXED") and consequently by SETVERB("DECIMALS"). If SETVERB("FIXED") is True, numbers are displayed with a fixed decimal position. The number of decimal positions is defined by SETVERB("DECIMALS"). For that reason, the default is SETVERB("FIXED", .F.) and SETVERB("DECIMALS", 2), that is, no fixed decimal position, but if they will be activated, the default is two decimal digits.

Logical

The logical data type identifies Boolean values.

Logical constants are:

'**.T.**'	True.
'**.F.**'	False.

When editing a logical field, inputs may be:

y, Y, t, T	for True
n, N, f, F	for False

The following table shows all operations available inside nB for logical data types. These operations act on one or more logical expressions and the result is not necessarily a logical data type.

.AND.	And.
.OR.	Or.
.NOT. or !	Negate.
==	Compare for equity.
!=, <>, or #	Compare for inequity.

Comparing two logical values, False ('**.F.**') is always less than True ('**.T.**').

NIL

NIL is not properly a data type, it represent the value of an uninitialised variable.

Inside nB (like what it happens inside Clipper), variables are not declared with the data type that they will contain. This means that a variable can contain any kind of data. In fact, nB variables are pointer to data and a pointer to "nothing" is NIL.

NIL may be used as constant for assignment or comparing purpose:

```
NIL
```

Fields (database fields) cannot contain NIL.

The following table shows all operations available inside nB for the NIL data type. Except for these operations, attempting to operate on a NIL results in a runtime error.

==	Compare for equity.
!=, <>, #	Compare for inequity.
<	Compare for less than.
<=	Compare for less than or equal.
>	Compare for greater than.
>=	Compare for greater than or equal.
:=	In line assign.
EMPTY()	Test for NIL.
VALTYPE()	Evaluates data type directly.

For the purpose of comparison, NIL is the only value that is equal to NIL. All other values are greater than NIL.

Variables are created inside nB with MEMPUBLIC(). This function creates variables which will be automatically initialised to NIL.

Array

The array data type identifies a collection of related data items that share the same name. Each value in an array is referred to as an element.

Array elements can be of any data type except memo (memo is available only inside database fields). For example the first element can be a character string, the second a number, the third a date and so on. Arrays can contain other arrays and code blocks as elements.

The variable containing the array does not contains the entire array, but the reference to it.

When the NIL type was described, it was cleared that variables doesn't contains real data, but pointer to data. But this happens in a transparent way, that is that when the a variable is assigned to another (for example A := B) the variable receiving the assignment will receive a pointer to a new copy of the source data. This is not the same with arrays: assigning to a variable an array, will assign to that variable a pointer to the same source array and not to a new copy of it.

If arrays are to be duplicated, the ACLONE() function is to be used.

An array constant may be expressed using curly brackets {}. See the examples below.

```
A := { "first_element", "second_element", "third_element" }
```

With this example, the variable A contain the reference to an array with three element containing character string.

```
A[1] == "first_element"

A[2] == "second_element"

A[3] == "third_element"
```

Arrays may contain also no element: empty array and may be expressed as:

```
{}
```

The array element is identified by a number enclosed with square brackets, following the variable name containing the reference to the array. The first array element is one.

If an array contains arrays, we obtain a multidimensional array. For example:

```
A := { { 1, 2 }, { 3, 4 }, { 5, 6 } }
```

is equivalent to the following table.

1	2
3	4
5	6

With this example, the variable A contain the reference to a bidimensional array containing numbers.

A[1,1] or A[1][1] contains 1

A[1,2] or A[1][2] contains 2

A[2,1] or A[2][1] contains 3

and so on.

As arrays may contain mixed data, it is the user who have to handle correctly the element numbers. For example:

```
A := { "hello", { 3, 4 }, 1234 }

A[1] == "hello"

A[2] == reference to { 3, 4 }

A[3] == 1234
```

A[2,1] or A[2][1] contains 3

A[2,2] or A[2][2] contains 4

A[1,1] is an error!

The following table shows all operations available inside nB for arrays.

:=	In line assign.
AADD()	Add dynamically an element to an array.
ACLONE()	Create a copy of an array.
ACOPY()	Copy element by element an array to another.
ADEL()	Delete one element inside an array.
AFILL()	Fill all array elements with a value.
AINS()	Insert an element inside an array.
ARRAY()	Creates an array of empty elements.
ASCAN()	Scan the array elements.
ASIZE()	Resize an array.
ASORT()	Sort the array elements.
EMPTY()	Test for no elements.
VALTYPE()	Evaluates data type directly.

Code block

The code block data type identifies a small piece of executable program code.

A code block is something like a little user defined function where only a sequence of functions or assignments may appear: no loops, no IF ELSE END.

A code block may receive argument and return a value after execution, just like a function.

The syntax is:

```
{ | [argument_list] | exp_list }
```

That is: the ***argument_list*** is optional; the ***exp_list*** may contain one or more expressions separated with a comma.

For example, calling the following code block will give the string "hello world" as result.

```
{ || "hello world" }
```

The following code block require a numeric argument an returns the number passed as argument incremented:

```
{ | n | n+1 }
```

The following code block requires two numeric arguments and returns the sum of the two square radix:

```
{ | nFirst, nSecond | SQRT(nFirst) + SQRT(nSecond) }
```

But code blocks may contains more expressions and the result of the execution of the code block is the result of the last expression.

The following code block executes in sequence some functions and give ever "hello world" as a result.

```
{ | a, b | functionOne(a), functionTwo(b), "hello world" }
```

To start the execution of a code block a function is used: EVAL()

For example, a code block is assigned to a variable and then executed.

```
B := { || "hello world" }
```

EVAL(B) == "hello world"

Another example with a parameter.

```
B := { | n | n+1 }
```

EVAL(B, 1) == 2

Another example with two parameters.

```
B := { | nFirst, nSecond | SQRT(nFirst) + SQRT(nSecond) }
```

EVAL(B, 2, 4) == 20

And so on.

The following table shows some operations available inside nB for code blocks: many functions use code blocks as argument.

:=	In line assign.
AEVAL()	Evaluate (execute) a code block for each element in an array.
BCOMPILE()	Convert (compile) a character string into a code block.
DBEVAL()	Evaluate (execute) a code block for each record in the active alias.
EVAL()	Evaluate a code block once.
VALTYPE()	Evaluates data type directly.

Operators

Here is a list with a brief description of the operators available inside nB.

```
cString1 $ cString2
```

Substring comparison.

If ***cString1*** is contained inside ***cString2*** the result is true ('`.T.`').

```
nNumber1 % nNumber2
```

Modulus.

The result is the remainder of ***nNumber1*** divided by ***nNuber2***.

```
()
```

Function or grouping indicator.

```
nNumber1 * nNumber2
```

Multiplication.

```
nNumber1 ** nNumber2
nNumber1 ^ nNumber2
```

Exponentiation.

```
nNumber1 + nNumber2
dDate + nNumber
```

Addition, unary positive.

```
cString1 + cString2
```

String concatenation.

The result is a string beginning with the content of ***cString1*** and following with the content of ***cString2***.

```
nNumber1 - nNumber2
dDate1 - dDate2
dDate - nNumber
```

Subtraction, unary negative.

```
cString1 - cString2
```

String concatenation.

The result is a string containing ***cString1*** after trimming trailing blanks and ***cString2***.

```
idAlias->idField
FIELD->idVar
MEMVAR->idVar
```

Alias assignment.

The alias operator implicitly SELECTs the ***idAlias*** before evaluating ***idField***. When the evaluation is complete, the original work area is SELECTed again.


```
lCondition1 .AND. lCondition2
```

Logical AND.

```
.NOT. lCondition
```

Logical NOT.

```
lCondition1 .OR. lCondition2
```

Logical OR.

```
nNumber1 / nNumber2
```

Division.

```
object:message[(argument list)]
```

Send.

```
idVar := exp
```

Inline assign.

```
exp1 <= exp2
```

Less than or equal.

```
exp1 <> exp2
```

Not equal.

```
exp1 = exp2
```

Equal.

```
exp1 == exp2
```

Exactly equal.

```
exp1 > exp2
```

Greater than.

```
exp1 >= exp2
```

Greater than or equal.

```
@idVar
```

Pass-by-reference.

```
[]
aArray[nSubscript, ...]
aArray[nSubscript1][nSubscript2] ...
```

Array element indicator.

Delimiters

Here is the delimiter list recognised from nB.

```
{ exp_list }
```

Literal array delimiters.

```
{ |param_list| exp_list }
```

Code block delimiters.

```
"cString"
'cString'
[cString]
```

String delimiters.

Code blocks

A code block is a sequence of function, assignments and constant like the following:

```
sqrt(10)
nResult := 10 * nIndex
```

Suppose that the above sequence of operations has a meaning for you. We want to create a box containing this sequence of operation. This box is contained inside a variable:

```
bBlackBox := { || sqrt(10), nResult := 10 * nIndex }
```

Note the comma used as separator.

Now ***bBlackBox*** contains the small sequence seen before. To execute this sequence, the function EVAL() is used:

```
EVAL(bBlackBox)
```

The execution of the code block gives a result: the value of the last operation contained inside the code block. In this case it is the result of 10****nIndex***. For that reason, if the execution of the code block must give a fixed result, it can terminate with a constant.

A code block may receive parameters working like a function. Try to imagine that we need to do the following.

```
function multiply( nVar1, nVar2 )
     return nVar * nVar2
endfunction
```

A code block that does the same is:

```
bMultiply := { | nVar1, nVar2 | nVar1 * nVar2 }
```

To evaluate it, for example trying to multiply 10 * 5:

```
nResult := EVAL( bMultiply, 10, 5 )
```

and ***nResult*** will contain 50.

Standard functions

With nB all Clipper standard functions may be used. Here follows a short description.

AADD()

Array add

AADD(***aTarget***, ***expValue***) ⇒ ***Value***

aTarget	is the array to add a new element to.
expValue	is the value assigned to the new element.

It increases the actual length of the target array by one. The newly created array element is assigned the value specified by ***expValue***.

ABS()

Absolute

ABS(***nExp***) ⇒ ***nPositive***

nExp	is the numeric expression to evaluate.

ABS() returns a number representing the absolute value of its argument.

ACLONE()

Array clone

ACLONE(***aSource***) ⇒ ***aDuplicate***

aSource	is the array to duplicate.

ACLONE() returns a duplicate of ***aSource***.

ACOPY()

Array copy

ACOPY(***aSource***, ***aTarget***, [***nStart***], [***nCount***], [***nTargetPos***]) ⇒ ***aTarget***

aSource	is the array to copy elements from.
aTarget	is the array to copy elements to.
nStart	is the starting element position in the ***aSource*** array. If not specified, the default value is one.
nCount	is the number of elements to copy from the ***aSource*** array beginning at the ***nStart*** position. If ***nCount*** is not specified, all elements in ***aSource*** beginning with the starting element are copied.
nTargetPos	is the starting element position in the ***aTarget*** array to receive elements from ***aSource***. If not specified, the default value is one.

ACOPY() is an array function that copies elements from the ***aSource*** array to the ***aTarget*** array. The ***aTarget*** array must already exist and be large enough to hold the copied elements.

ADEL()

Array delete

ADEL(***aTarget***, ***nPosition***) ⇒ ***aTarget***

aTarget	is the array to delete an element from.
nPosition	is the position of the target array element to delete.

ADEL() is an array function that deletes an element from an array. The contents of the specified array element is lost, and all elements from that position to the end of the array are shifted up one element. The last element in the array becomes NIL.

AEVAL()

Array evaluation

AEVAL(***aArray***, ***bBlock***, [***nStart***], [***nCount***]) ⇒ ***aArray***

aArray	is the array to be evaluated.
bBlock	is a code block to execute for each element encountered.
nStart	is the starting element. If not specified, the default is element one.
nCount	is the number of elements to process from ***nStart***. If not specified, the default is all elements to the end of the array.

AEVAL() is an array function that evaluates a code block once for each element of an array, passing the element value and the element index as block parameters. The return value of the block is ignored. All elements in ***aArray*** are processed unless either the ***nStart*** or the ***nCount*** argument is specified.

AFILL()

Array fill

AFILL(***aTarget***, ***expValue***, [***nStart***], [***nCount***]) ⇒ ***aTarget***

aTarget	is the array to fill.
expValue	is the value to place in each array element. It can be an expression of any valid data type.
nStart	is the position of the first element to fill. If this argument is omitted, the default value is one.
nCount	is the number of elements to fill starting with element ***nStart***. If this argument is omitted, elements are filled from the starting element position to the end of the array.

AFILL() is an array function that fills the specified array with a single value of any data type (including an array, code block, or NIL) by assigning ***expValue*** to each array element in the specified range.

AINS()

Array insert

```
AINS(aTarget, nPosition) ⇒ aTarget
```

aTarget	is the array into which a new element will be inserted.
nPosition	is the position at which the new element will be inserted.

AINS() is an array function that inserts a new element into a specified array. The newly inserted element is NIL data type until a new value is assigned to it. After the insertion, the last element in the array is discarded, and all elements after the new element are shifted down one position.

ALERT()

```
ALERT( cMessage, [aOptions] ) ⇒ nChoice
```

cMessage	is the message text displayed, centered, in the alert box. If the message contains one or more semicolons, the text after the semicolons is centered on succeeding lines in the dialog box.
aOptions	defines a list of up to 4 possible responses to the dialog box.

ALERT() returns a numeric value indicating which option was chosen. If the Esc key is pressed, the value returned is zero. The ALERT() function creates a simple modal dialog. The user can respond by moving a highlight bar and pressing the Return or Space-Bar keys, or by pressing the key corresponding to the first letter of the option. If ***aOptions*** is not supplied, a single "Ok" option is presented.

ALIAS()

```
ALIAS([nWorkArea]) ⇒ cAlias
```

nWorkArea	is any work area number.

ALIAS() returns the alias of the specified work area as a character string. If ***nWorkArea*** is not specified, the alias of the current work area is returned. If there is no database file in USE for the specified work area, ALIAS() returns a null string ("").

ALLTRIM()

```
ALLTRIM(cString) ⇒ cTrimmedString
```

cString	is the character expression to trim.

ALLTRIM() returns a character string with leading and trailing spaces removed.

ARRAY()

```
ARRAY(nElements [, nElements...]) ⇒ aArray
```

nElements	is the number of elements in the specified dimension.

ARRAY() is an array function that returns an uninitialized array with the specified number of elements and dimensions.

ASC()

ASCII

```
ASC(cExp) ⇒ nCode
```

cExp	is the character expression to convert to a number.

ASC() returns an integer numeric value in the range of zero to 255 , representing the ASCII value of ***cExp***.

ASCAN()

Array scan

```
ASCAN(aTarget, expSearch,
      [nStart], [nCount]) ⇒ nStoppedAt
```

aTarget	is the array to scan.
expSearch	is either a simple value to scan for, or a code block. If ***expSearch*** is a simple value it can be character, date, logical, or numeric type.
nStart	is the starting element of the scan. If this argument is not specified, the default starting position is one.
nCount	is the number of elements to scan from the starting position. If this argument is not specified, all elements from the starting element to the end of the array are scanned.

ASCAN() returns a numeric value representing the array position of the last element scanned. If ***expSearch*** is a simple value, ASCAN() returns the position of the first matching element, or zero if a match is not found. If ***expSearch*** is a code block, ASCAN() returns the position of the element where the block returned true ('`.T.`').

ASIZE()

Array size

```
ASIZE(aTarget, nLength) ⇒ aTarget
```

aTarget	is the array to grow or shrink.
nLength	is the new size of the array.

ASIZE() is an array function that changes the actual length of the ***aTarget*** array. The array is shortened or lengthened to match the specified length. If the array is shortened, elements at the end of the array are lost. If the array is lengthened, new elements are added to the end of the array and assigned NIL.

ASORT()

Array sort

```
ASORT(aTarget, [nStart],
      [nCount], [bOrder]) ⇒ aTarget
```

aTarget	is the array to sort.

nStart	is the first element of the sort. If not specified, the default starting position is one.
nCount	is the number of elements to sort. If not specified, all elements in the array beginning with the starting element are sorted.
bOrder	is an optional code block used to determine sorting order. If not specified, the default order is ascending.

ASORT() is an array function that sorts all or part of an array containing elements of a single data type. Data types that can be sorted include character, date, logical, and numeric. If the ***bOrder*** argument is not specified, the default order is ascending. Each time the block is evaluated, two elements from the target array are passed as block parameters. The block must return true (‘**.T.**’) if the elements are in sorted order.

AT()

«

`AT(`***cSearch***`, `***cTarget***`) ⇒ `***nPosition***

cSearch	is the character substring for which to search.
cTarget	is the character string to search.

AT() returns the position of the first instance of ***cSearch*** within ***cTarget*** as an integer numeric value. If ***cSearch*** is not found, AT() returns zero.

AT() is a character function used to determine the position of the first occurrence of a character substring within another string.

ATAIL()

«

Array TAIL

`ATAIL(`***aArray***`) ⇒ `***Element***

aArray	is the array.

ATAIL() is an array function that returns the highest numbered element of an array. It can be used in applications as shorthand for ***aArray***[LEN(***aArray***)] when you need to obtain the last element of an array.

BIN2I()

«

Binary to integer

`BIN2I(`***cSignedInt***`) ⇒ `***nNumber***

cSignedInt	is a character string in the form of a 16-bit signed integer number--least significant byte first.

BIN2I() returns an integer obtained converting the first two byte contained inside ***cSignedInt***.

BIN2L()

«

Binary to long

`BIN2L(`***cSignedInt***`) ⇒ `***nNumber***

cSignedInt	is a character string in the form of a 32-bit signed integer number--least significant byte first.

BIN2L() returns an integer obtained from the first tour characters contained in ***cSignedInt***.

BIN2W()

«

Binary to word

`BIN2W(`***cUnsignedInt***`) ⇒ `***nNumber***

cUnsignedInt	is a character string in the form of a 16-bit unsigned integer number--least significant byte first.

BIN2W() returns an integer obtained from the first two characters contained in ***cSignedInt***.

BOF()

«

Begin of file

`BOF() ⇒ `***lBoundary***

BOF() returns true (‘**.T.**’) after an attempt to SKIP backward beyond the first logical record in a database file; otherwise, it returns false (‘**.F.**’). If there is no database file open in the current work area, BOF() returns false (‘**.F.**’). If the current database file contains no records, BOF() returns true (‘**.T.**’).

CDOW()

«

Character day of week

`CDOW(`***dExp***`) ⇒ `***cDayName***

dExp	is the date value to convert.

CDOW() returns the name of the day of the week as a character string. The first letter is uppercase and the rest of the string is lowercase. For a null date value, CDOW() returns a null string ("").

CHR()

«

Character

`CHR(`***nCode***`) ⇒ `***cChar***

nCode	is an ASCII code in the range of zero to 255.

CHR() returns a single character value whose ASCII code is specified by ***nCode***.

CMONTH()

«

Character month

`CMONTH(`***dDate***`) ⇒ `***cMonth***

dDate	is the date value to convert.

CMONTH() returns the name of the month as a character string from a date value with the first letter uppercase and the rest of the string lowercase. For a null date value, CMONTH() returns a null string ("").

COL()

«

Column

`COL() ⇒ `***nCol***

COL() is a screen function that returns the current column position of the cursor. The value of COL() changes whenever the cursor position changes on the screen.

COLORSELECT()

```
COLORSELECT( nColorIndex ) ⇒ NIL
```

nColorIndex	is a number corresponding to the ordinal positions in the current list of color attributes, as set by SETCOLOR().

COLORSELECT() activates the specified color pair from the current list of color attributes (established by SETCOLOR()).

CTOD()

Character to date

```
CTOD( cDate ) ⇒ dDate
```

cDate	is a character string consisting of numbers representing the month, day, and year separated by any character other than a number. The month, day, and year digits must be specified in accordance with the SET DATE format. If the century digits are not specified, the century is determined by the rules of SET EPOCH.

CTOD() returns a date value. If ***cDate*** is not a valid date, CTOD() returns an empty date.

CURDIR()

Current directory

```
CURDIR( [cDrivespec] ) ⇒ cDirectory
```

cDrivespec	specifies the letter of the disk drive to query. If not specified, the default is the current DOS drive.

CURDIR() returns the current DOS directory of the drive specified by ***cDrivespec*** as a character string without either leading or trailing backslash (\) characters.

DATE()

```
DATE() ⇒ dSystemDate
```

DATE() returns the system date as a date value.

DAY()

```
DAY( dDate ) ⇒ nDay
```

dDate	is a date value to convert.

DAY() returns the day number from ***dDate***.

DBAPPEND()

```
DBAPPEND( [lReleaseRecLocks] ) ⇒ NIL
```

lReleaseRecLocks	is a logical data type that if true (‘**.T.**’), clears all pending record locks, then appends the next record. If ***lReleaseRecLocks*** is false (‘**.F.**’), all pending record locks are maintained and the new record is added to the end of the Lock List. The default value of ***lReleaseRecLocks*** is true (‘**.T.**’).

DBAPPEND() adds a new empty record to the active alias.

DBCLEARFILTER()

```
DBCLEARFILTER() ⇒ NIL
```

DBCLEARFILTER() clears the logical filter condition, if any, for the current work area.

DBCLEARINDEX()

```
DBCLEARINDEX() ⇒ NIL
```

DBCLEARINDEX() closes any active indexes for the active alias.

DBCLEARRELATION()

```
DBCLEARRELATION() ⇒ NIL
```

DBCLEARRELATION() clears any active relations for the active alias.

DBCLOSEALL()

```
DBCLOSEALL() ⇒ NIL
```

DBCLOSEALL() releases all occupied work areas from use. It is equivalent to calling DBCLOSEAREA() on every occupied work area.

Attention: DBCLOSEALL() cannot be used inside a "compiled" macro as this will stop the macro execution. In substitution, DBCLOSE() should be used.

DBCLOSEAREA()

```
DBCLOSEAREA() ⇒ NIL
```

DBCLOSEAREA() releases the current work area from use.

DBCOMMIT()

```
DBCOMMIT() ⇒ NIL
```

DBCOMMIT() causes all updates to the current work area to be written to disk. All updated database and index buffers are written to DOS and a DOS COMMIT request is issued for the database (.dbf) file and any index files associated with the work area. Inside a network environment, DBCOMMIT() makes database updates visible to other processes. To insure data integrity, issue DBCOMMIT() before an UNLOCK operation.

DBCOMMITALL()

```
DBCOMMITALL() ⇒ NIL
```

DBCOMMITALL() causes all pending updates to all work areas to be written to disk. It is equivalent to calling DBCOMMIT() for every occupied work area.

DBCREATE()

```
DBCREATE( cDatabase, aStruct, [cDriver] ) ⇒ NIL
```

cDatabase	is the name of the new database file, with an optional drive and directory, specified as a character string. If specified without an extension (.dbf) is assumed.

aStruct	is an array that contains the structure of ***cDatabase*** as a series of subarrays, one per field. Each subarray contains the definition of each field's attributes and has the following structure: ***aStruct[n][1]*** == ***cName*** ***aStruct[n][2]*** == ***cType*** ***aStruct[n][3]*** == ***nLength*** ***aStruct[n][4]*** == ***nDecimals***
cDriver	specifies the replaceable database driver (RDD) to use to process the current work area. ***cDriver*** is name of the RDD specified as a character expression.

DBCREATE() is a database function that creates a database file from an array containing the structure of the file.

DBCREATEINDEX()

«

```
DBCREATEINDEX(cIndexName, cKeyExpr, bKeyExpr, [lUnique])
      ⇒ NIL
```

cIndexName	is a character value that specifies the filename of the index file (order bag) to be created.
cKeyExpr	is a character value that expresses the index key expression in textual form.
bKeyExpr	is a code block that expresses the index key expression in executable form.
lUnique	is an optional logical value that specifies whether a unique index is to be created. If ***lUnique*** is omitted, the current global _SET_UNIQUE setting is used.

DBCREATEINDEX() creates an index for the active alias. If the alias has active indexes, they are closed.

DBDELETE()

«

```
DBDELETE() ⇒ NIL
```

DBDELETE() marks the current record as deleted (*). Records marked for deletion can be filtered using SET DELETED or removed from the file using the PACK command.

DBEVAL()

«

DB evaluate

```
DBEVAL(bBlock,
       [bForCondition],
       [bWhileCondition],
       [nNextRecords],
       [nRecord],
       [lRest]) ⇒ NIL
```

bBlock	is a code block to execute for each record processed.
bForCondition	the FOR condition expressed as code block.
bWhileCondition	the WHILE condition expressed as code block.
nNextRecords	is an optional number that specifies the number of records to process starting with the current record. It is the same as the NEXT clause.
nRecord	is an optional record number to process. If this argument is specified, ***bBlock*** will be evaluated for the specified record. This argument is the same as the RECORD clause.
lRest	is an optional logical value that determines whether the scope of DBEVAL() is all records, or, starting with the current record, all records to the end of file.

DBEVAL() is a database function that evaluates a single block for each record within the active alias.

DBFILTER()

«

```
DBFILTER() ⇒ cFilter
```

BFILTER() returns the filter condition defined in the current work area as a character string. If no FILTER has been SET, DBFILTER() returns a null string ("").

DBGOBOTTOM()

«

```
DBGOBOTTOM() ⇒ NIL
```

DBGOBOTTOM() moves to last logical record in the active alias.

DBGOTO()

«

```
DBGOTO(nRecordNumber) ⇒ NIL
```

nRecordNumber	is a numeric value that specifies the record number of the desired record.

DBGOTO() moves to the record whose record number is equal to ***nRecordNumber***. If no such record exists, the work area is positioned to LASTREC() + 1 and both EOF() and BOF() return true ('**.T.**').

DBGOTOP()

«

```
DBGOTOP() ⇒ NIL
```

DBGOTOP() moves to the first logical record in the current work area.

DBRECALL()

«

```
DBRECALL() ⇒ NIL
```

DBRECALL() causes the current record to be reinstated if it is marked for deletion.

DBREINDEX()

«

```
DBREINDEX() ⇒ NIL
```

DBREINDEX() rebuilds all active indexes associated with the active alias.

DBRELATION()

«

```
DBRELATION(nRelation) ⇒ cLinkExp
```

nRelation	is the position of the desired relation in the list of active alias relations.

DBRELATION() returns a character string containing the linking expression of the relation specified by ***nRelation***. If there is no RELATION SET for ***nRelation***, DBRELATION() returns a null string ("").

DBRLOCK()

«

DB record lock

```
DBRLOCK([nRecNo]) ⇒ lSuccess
```

nRecNo	is the record number to be locked. The default is the current record.

DBRLOCK() is a database function that locks the record identified by ***nRecNo*** or the current record.

DBRLOCKLIST()

```
DBRLOCKLIST() ⇒ aRecordLocks
```

DBRLOCKLIST() returns a one-dimensional array of the locked records in the active alias.

DBRSELECT()

DB relation select

```
DBRSELECT(nRelation) ⇒ nWorkArea
```

nRelation	is the position of the desired relation in the list of current work area relations.

DBRSELECT() returns the work area number of the relation specified by ***nRelation*** as an integer numeric value. If there is no RELATION SET for ***nRelation***, DBRSELECT() returns zero.

DBRUNLOCK()

DB relation unlock

```
DBRUNLOCK([nRecNo]) ⇒ NIL
```

nRecNo	is the record number to be unlocked. The default is all previously locked records.

DBRUNLOCK() is a database function that unlocks the record identified by ***nRecNo*** or all locked records.

DBSEEK()

```
DBSEEK(expKey, [lSoftSeek]) ⇒ lFound
```

expKey	is a value of any type that specifies the key value associated with the desired record.
lSoftSeek	is an optional logical value that specifies whether a soft seek is to be performed. This determines how the work area is positioned if the specified key value is not found. If ***lSoftSeek*** is omitted, the current global _SET_SOFTSEEK setting is used.

DBSEEK() returns true (‘**.T.**’) if the specified key value was found; otherwise, it returns false (‘**.F.**’).

DBSELECTAREA()

```
DBSELECTAREA(nArea | cAlias) ⇒ NIL
```

nArea	is a numeric value between zero and 250, inclusive, that specifies the work area being selected.
cAlias	is a character value that specifies the alias of a currently occupied work area being selected.

DBSELECTAREA() causes the specified work area to become the current work area. All subsequent database operations will apply to this work area unless another work area is explicitly specified for an operation.

DBSETDRIVER()

```
DBSETDRIVER([cDriver]) ⇒ cCurrentDriver
```

cDriver	is an optional character value that specifies the name of the database driver that should be used to activate and manage new work areas when no driver is explicitly specified.

DBSETDRIVER() returns the name of the current default driver.

DBSETFILTER()

```
DBSETFILTER(bCondition, [cCondition]) ⇒ NIL
```

bCondition	is a code block that expresses the filter condition in executable form.
cCondition	is a character value that expresses the filter condition in textual form. If ***cCondition*** is omitted, the DBSETFILTER() function will return an empty string for the work area.

DBSETFILTER() sets a logical filter condition for the current work area. When a filter is set, records which do not meet the filter condition are not logically visible. That is, database operations which act on logical records will not consider these records. The filter expression supplied to DBSETFILTER() evaluates to true (‘**.T.**’) if the current record meets the filter condition; otherwise, it should evaluate to false (‘**.F.**’).

DBSETINDEX()

```
DBSETINDEX(cOrderBagName) ⇒ NIL
```

cOrderBagName	is a character value that specifies the filename of the index file (index bag) to be opened.

DBSETINDEX() is a database function that adds the contents of an Order Bag into the Order List of the current work area. Any Orders already associated with the work area continue to be active. If the newly opened Order Bag is the only Order associated with the work area, it becomes the controlling Order; otherwise, the controlling Order remains unchanged. If the Order Bag contains more than one Order, and there are no other Orders associated with the work area, the first Order in the new Order Bag becomes the controlling Order.

DBSETORDER()

```
DBSETORDER(nOrderNum) ⇒ NIL
```

nOrderNum	is a numeric value that specifies which of the active indexes is to be the controlling index.

DBSETORDER() controls which of the active alias’ active indexes is the controlling index.

DBSETRELATION()

```
DBSETRELATION(nArea | cAlias, bExpr, [cExpr]) ⇒ NIL
```

nArea	is a numeric value that specifies the work area number of the child work area.
cAlias	is a character value that specifies the alias of the child work area.
bExpr	is a code block that expresses the relational expression in executable form.
cExpr	is an optional character value that expresses the relational expression in textual form. If ***cExpr*** is omitted, the DBRELATION() function returns an empty string for the relation.

DBSETRELATION() relates the work area specified by ***nArea*** or ***cAlias*** (the child work area), to the current work area (the parent work area). Any existing relations remain active.

DBSKIP()

«

```
DBSKIP( [nRecords] ) ⇒ NIL
```

nRecords	is the number of logical records to move, relative to the current record. A positive value means to skip forward, and a negative value means to skip backward. If ***nRecords*** is omitted, a value of 1 is assumed.

DBSKIP() moves either forward or backward relative to the current record. Attempting to skip forward beyond the last record positions the work area to LASTREC() + 1 and EOF() returns true (‘**.T.**’). Attempting to skip backward beyond the first record positions the work area to the first record and BOF() returns true (‘**.T.**’).

DBSTRUCT()

«

```
DBSTRUCT() ⇒ aStruct
```

DBSTRUCT() returns the structure of the current database file in an array whose length is equal to the number of fields in the database file. Each element of the array is a subarray containing information for one field. The subarrays have the following format:

```
aStruct[n][1] == cName
aStruct[n][2] == cType
aStruct[n][3] == nLength
aStruct[n][4] == nDecimals
```

If there is no database file in USE in the current work area, DBSTRUCT() returns an empty array ({}).

DBUNLOCK()

«

```
DBUNLOCK() ⇒ NIL
```

DBUNLOCK() releases any record or file locks obtained by the current process for the current work area. DBUNLOCK() is only meaningful on a shared database in a network environment.

DBUNLOCKALL()

«

```
DBUNLOCKALL() ⇒ NIL
```

DBUNLOCKALL() releases any record or file locks obtained by the current process for any work area. DBUNLOCKALL() is only meaningful on a shared database in a network environment.

DBUSEAREA()

«

```
DBUSEAREA( [lNewArea], [cDriver], cName, [xcAlias],
     [lShared], [lReadonly] ) ⇒ NIL
```

lNewArea	is an optional logical value. A value of true (‘**.T.**’) selects the lowest numbered unoccupied work area as the current work area before the use operation. If ***lNewArea*** is false (‘**.F.**’) or omitted, the current work area is used; if the work area is occupied, it is closed first.
cDriver	is an optional character value. If present, it specifies the name of the database driver which will service the work area. If ***cDriver*** is omitted, the current default driver is used.

cName	specifies the name of the database (.dbf) file to be opened.
xcAlias	is an optional character value. If present, it specifies the alias to be associated with the work area. The alias must constitute a valid identifier. A valid ***xcAlias*** may be any legal identifier (i.e., it must begin with an alphabetic character and may contain numeric or alphabetic characters and the underscore). If ***xcAlias*** is omitted, a default alias is constructed from ***cName***.
lShared	is an optional logical value. If present, it specifies whether the database (.dbf) file should be accessible to other processes on a network. A value of true (‘**.T.**’) specifies that other processes should be allowed access; a value of false (‘**.F.**’) specifies that the current process is to have exclusive access. If ***lShared*** is omitted, the current global _SET_EXCLUSIVE setting determines whether shared access is allowed.
lReadonly	is an optional logical value that specifies whether updates to the work area are prohibited. A value of true (‘**.T.**’) prohibits updates; a value of false (‘**.F.**’) permits updates. A value of true (‘**.T.**’) also permits read-only access to the specified database (.dbf) file. If ***lReadonly*** is omitted, the default value is false (‘**.F.**’).

DBUSEAREA() opens the specified database (.DBF).

DBDELETE()

«

```
DELETED() ⇒ lDeleted
```

DELETED() returns true (‘**.T.**’) if the current record is marked for deletion; otherwise, it returns false (‘**.F.**’). If there is no database file in USE in the current work area, DELETED() returns false (‘**.F.**’).

DESCEND()

«

```
DESCEND(exp) ⇒ ValueInverted
```

exp	is any valid expression of character, date, logical, or numeric type.

DESCEND() returns an inverted expression of the same data type as the ***exp***, except for dates which return a numeric value. A DESCEND() of CHR(0) always returns CHR(0).

DEVOUT()

«

Device output

```
DEVOUT(exp, [cColorString]) ⇒ NIL
```

exp	is the value to display.
cColorString	is an optional argument that defines the display color of ***exp***.

DEVOUT() is a full-screen display function that writes the value of a single expression to the current device at the current cursor or printhead position.

DEVOUTPICT()

«

Device output picture

```
DEVOUTPICT(exp, cPicture, [cColorString]) ⇒ NIL
```

exp	is the value to display.
cPicture	defines the formatting control for the display of ***exp***.
cColorString	is an optional argument that defines the display color of ***exp***.

DEVOUTPICT() is a full-screen display function that writes the value of a single expression to the current device at the current cursor or printhead position.

DEVPOS()

Device position

```
DEVPOS(nRow, nCol) ⇒ NIL
```

nRow, ***nCol***	are the new row and column positions of the cursor or printhead.

DEVPOS() is an environment function that moves the screen or printhead depending on the current DEVICE.

DIRECTORY()

```
DIRECTORY(cDirSpec, [cAttributes]) ⇒ aDirectory
```

cDirSpec	identifies the drive, directory and file specification for the directory search. Wildcards are allowed in the file specification. If ***cDirSpec*** is omitted, the default value is *.*.
cAttributes	specifies inclusion of files with special attributes in the returned information. ***cAttributes*** is a string containing one or more of the following characters: H Include hidden files S Include system files D Include directories V Search for the DOS volume label only Normal files are always included in the search, unless you specify V.

DIRECTORY() returns an array of subarrays, with each subarray containing information about each file matching ***cDirSpec***. The subarray has the following structure:

```
aDirectory[n][1] == cName
aDirectory[n][2] == cSize
aDirectory[n][3] == dDate
aDirectory[n][4] == cTime
aDirectory[n][5] == cAttributes
```

If no files are found matching ***cDirSpec*** or if ***cDirSpec*** is an illegal path or file specification, DIRECTORY() returns an empty ({}) array.

DISKSPACE()

```
DISKSPACE([nDrive]) ⇒ nBytes
```

nDrive	is the number of the drive to query, where one is drive A, two is B, three is C, etc. The default is the current DOS drive if ***nDrive*** is omitted or specified as zero.

DISKSPACE() returns the number of bytes of empty space on the specified disk drive as an integer numeric value.

DISPBOX()

Display box

```
DISPBOX(nTop, nLeft, nBottom, nRight,
    [cnBoxString], [cColorString]) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom***, ***nRight***	define the coordinates of the box.
cnBoxString	is a numeric or character expression that defines the border characters of the box. If specified as a numeric expression, a value of 1 displays a single-line box and a value of 2 displays a double-line box. All other numeric values display a single-line box. If ***cnBoxString*** is a character expression, it specifies the characters to be used in drawing the box. This is a string of eight border characters and a fill character.
cColorString	defines the display color of the box that is drawn.

DISPBOX() is a screen function that draws a box at the specified display coordinates in the specified color.

DISPOUT()

Display out

```
DISPOUT(exp, [cColorString]) ⇒ NIL
```

exp	is the value to display.
cColorString	is an optional argument that defines the display color of ***exp***.
cColorString	is a character expression containing the standard color setting.

DISPOUT() is a simple output function that writes the value of a single expression to the display at the current cursor position. This function ignores the SET DEVICE setting; output always goes to the screen.

DOW()

Day of week

```
DOW(dDate) ⇒ nDay
```

dDate	is a date value to convert.

DOW() returns the day of the week as a number between zero and seven. The first day of the week is one (Sunday) and the last day is seven (Saturday). If ***dDate*** is empty, DOW() returns zero.

DTOC()

Date to character

```
DTOC(dDate) ⇒ cDate
```

dDate	is the date value to convert.

DTOC() returns a character string representation of a date value. The return value is formatted in the current date format. A null date returns a string of spaces equal in length to the current date format.

DTOS()

Date to sort

```
DTOS(dDate) ⇒ cDate
```

dDate	is the date value to convert.

DTOS() returns a character string eight characters long in the form, yyyymmdd. When ***dDate*** is a null date (CTOD("")), DTOS() returns a string of eight spaces.

EMPTY()

«

`EMPTY(`***exp***`)` ⇒ ***lEmpty***

exp	is an expression of any data type.

EMPTY() returns true ('`.T.`') if the expression results in an empty value; otherwise, it returns false ('`.F.`'):

Array	{}
Character/Memo	Spaces, tabs, CR/LF, or ""
Numeric	0
Date	CTOD("")
Logical	'`.F.`'
NIL	NIL

EOF()

«

End of file

`EOF()` ⇒ ***lBoundary***

EOF() returns true ('`.T.`') when an attempt is made to move the record pointer beyond the last logical record in a database file; otherwise, it returns false ('`.F.`'). If there is no database file open in the current work area, EOF() returns false ('`.F.`'). If the current database file contains no records, EOF() returns true ('`.T.`').

EVAL()

«

Code block evaluation

`EVAL(`***bBlock***`, [`***BlockArg_list***`])` ⇒ ***LastBlockValue***

bBlock	is the code block to evaluate.
BlockArg_list	is a list of arguments to send to the code block before it is evaluated.

To execute or evaluate a code block, call EVAL() with the block value and any parameters. The parameters are supplied to the block when it is executed. Code blocks may be a series of expressions separated by commas. When a code block is evaluated, the returned value is the value of the last expression in the block.

EXP()

«

Exponent

`EXP(`***nExponent***`)` ⇒ ***nAntilogarithm***

nExponent	is the natural logarithm for which a numeric value is to be calculated.

EXP() returns a numeric value that is equivalent to the value e raised to the specified power.

FCLOSE()

«

File close

`FCLOSE(`***nHandle***`)` ⇒ ***lError***

nHandle	is the file handle obtained previously from FOPEN() or FCREATE().

FCLOSE() is a low-level file function that closes binary files and forces the associated DOS buffers to be written to disk. If the operation fails, FCLOSE() returns false ('`.F.`'). FERROR() can then be used to determine the reason for the failure.

FCOUNT()

«

Field count

`FCOUNT()` ⇒ ***nFields***

FCOUNT() returns the number of fields in the database file in the active alias as an integer numeric value. If there is no database file open, FCOUNT() returns zero.

FCREATE()

«

Field create

`FCREATE(`***cFile***`, [`***nAttribute***`])` ⇒ ***nHandle***

cFile	is the name of the file to create. If the file already exists, its length is truncated to zero without warning.
nAttribute	is the binary file attribute, the default value is zero. ***nAttribute*** = 0 Normal (default) ***nAttribute*** = 1 Read-only ***nAttribute*** = 2 Hidden ***nAttribute*** = 4 System

FCREATE() returns the DOS file handle number of the new binary file in the range of zero to 65,535. If an error occurs, FCREATE() returns -1 and FERROR() is set to indicate an error code.

FERASE()

«

File erase

`FERASE(`***cFile***`)` ⇒ ***nSuccess***

cFile	is the name (with or without path) of the file to be deleted from disk.

FERASE() is a file function that deletes a specified file from disk. FERASE() returns -1 if the operation fails and zero if it succeeds.

FERROR()

«

File error

`FERROR()` ⇒ ***nErrorCode***

FERROR() returns the DOS error from the last file operation as an integer numeric value. If there is no error, FERROR() returns zero.

nErrorCode value	Meaning
0	Successful
2	File not found
3	Path not found
4	Too many files open
5	Access denied
6	Invalid handle
8	Insufficient memory
15	Invalid drive specified
19	Attempted to write to a write-protected disk
21	Drive not ready
23	Data CRC error
29	Write fault
30	Read fault
32	Sharing violation
33	Lock Violation

FERROR() is a low-level file function that indicates a DOS error after a file function is used.

FIELDBLOCK()

FIELDBLOCK(*cFieldName*) ⇒ *bFieldBlock*

cFieldName	is the name of the field to which the set-get block will refer.

FIELDBLOCK() returns a code block that, when evaluated, sets (assigns) or gets (retrieves) the value of the given field. If ***cFieldName*** does not exist in the current work area, FIELDBLOCK() returns NIL.

FIELDGET()

FIELDGET(*nField*) ⇒ *ValueField*

nField	is the ordinal position of the field in the record structure for the current work area.

FIELDGET() returns the value of the specified field. If ***nField*** does not correspond to the position of any field in the current database file, FIELDGET() returns NIL.

FIELDNAME()

FIELDNAME(*nPosition*) ⇒ *cFieldName*

nPosition	is the position of a field in the database file structure.

FIELDNAME() returns the name of the specified field as a character string. If ***nPosition*** does not correspond to an existing field in the current database file or if no database file is open in the current work area, FIELDNAME() returns a null string ("").

FIELDPOS()

Field position

FIELDPOS(*cFieldName*) ⇒ *nFieldPos*

cFieldName	is the name of a field in the current or specified work area.

FIELDPOS() returns the position of the specified field within the list of fields associated with the current or specified work area. If the current work area has no field with the specified name, FIELDPOS() returns zero.

FIELDPUT()

FIELDPUT(*nField*, *expAssign*) ⇒ *ValueAssigned*

nField	is the ordinal position of the field in the current database file.
expAssign	is the value to assign to the given field. The data type of this expression must match the data type of the designated field variable.

FIELDPUT() is a database function that assigns ***expAssign*** to the field at ordinal position ***nField*** in the current work area. This function allows you to set the value of a field using its position within the database file structure rather than its field name.

FIELDWBLOCK()

Field work area block

FIELDWBLOCK(*cFieldName*, *nWorkArea*) ⇒ *bFieldWBlock*

cFieldName	is the name of the field specified as a character string.
nWorkArea	is the work area number where the field resides specified as a numeric value.

FIELDWBLOCK() returns a code block that, when evaluated, sets (assigns) or gets (retrieves) the value of ***cFieldName*** in the work area designated by ***nWorkArea***. If ***cFieldName*** does not exist in the specified work area, FIELDWBLOCK() returns NIL.

FILE()

FILE(*cFilespec*) ⇒ *lExists*

cFilespec	is in the current default directory and path. It is a standard file specification that can include the wildcard characters * and ? as well as a drive and path reference.

FILE() returns true ('**.T.**') if there is a match for any file matching the ***cFilespec*** pattern; otherwise, it returns false ('**.F.**').

FLOCK()

File lock

FLOCK() ⇒ *lSuccess*

FLOCK() tries to lock the active alias and returns true ('**.T.**') if it succeeds; otherwise, it returns false ('**.F.**').

FOPEN()

File open

FOPEN(*cFile*, [*nMode*]) ⇒ *nHandle*

cFile	is the name of the file to open including the path if there is one.
nMode	is the requested DOS open mode indicating how the opened file is to be accessed. The open mode is composed of the sum of two elements: the Open mode and the Sharing mode. Open mode: 0 Open for reading (default) 1 Open for writing 2 Open for reading or writing Sharing mode: 0 Compatibility mode (default) 16 Exclusive use 32 Prevent others from writing 48 Prevent others from reading 64 Allow others to read or write

FOPEN() returns the file handle of the opened file in the range of zero to 65,535. If an error occurs, FOPEN() returns -1.

FOUND()

FOUND() ⇒ *lSuccess*

FOUND() returns true ('**.T.**') if the last search command was successful; otherwise, it returns false ('**.F.**').

FREAD()

File read

FREAD(*nHandle*, *@cBufferVar*, *nBytes*) ⇒ *nBytes*

nHandle	is the file handle obtained from FOPEN(), FCREATE(), or predefined by DOS.

cBufferVar	is the name of an existing and initialized character variable used to store data read from the specified file. The length of this variable must be greater than or equal to ***nBytes***. ***cBufferVar*** must be passed by reference and, therefore, must be prefaced by the pass-by-reference operator (@).
nBytes	is the number of bytes to read into the buffer.

FREAD() tries to read ***nBytes*** of the binary file ***nHandle*** inside ***cBufferVar***. It returns the number of bytes successfully read as an integer numeric value. A return value less than ***nBytes*** or zero indicates end of file or some other read error.

FREADSTR()

File read string

`FREADSTR(`***nHandle***`, `***nBytes***`) ⇒ `***cString***

nHandle	is the file handle obtained from FOPEN(), FCREATE(), or predefined by DOS.
nBytes	is the number of bytes to read, beginning at the current DOS file pointer position.

FREADSTR() returns a character string up to 65,535 (64K) bytes. A null return value ("") indicates an error or end of file. FREADSTR() is a low-level file function that reads characters from an open binary file beginning with the current DOS file pointer position. Characters are read up to ***nBytes*** or until a null character (CHR(0)) is encountered. All characters are read including control characters except for CHR(0). The file pointer is then moved forward ***nBytes***. If ***nBytes*** is greater than the number of bytes from the pointer position to the end of the file, the file pointer is positioned to the last byte in the file.

FRENAME()

File rename

`FRENAME(`***cOldFile***`, `***cNewFile***`) ⇒ `***nSuccess***

cOldFile	is the name of the file to rename, including the file extension. A drive letter and/or path name may also be included as part of the filename.
cNewFile	is the new name of the file, including the file extension. A drive letter and/or path name may also be included as part of the name.

FRENAME() returns -1 if the operation fails and zero if it succeeds.

FSEEK()

File seek

`FSEEK(`***nHandle***`, `***nOffset***`, [`***nOrigin***`]) ⇒ `***nPosition***

nHandle	is the file handle obtained from FOPEN(), FCREATE(), or predefined by DOS.
nOffset	is the number of bytes to move the file pointer from the position defined by ***nOrigin***. It can be a positive or negative number. A positive number moves the pointer forward, and a negative number moves the pointer backward in the file.
nOrigin	defines the starting location of the file pointer before FSEEK() is executed. The default value is zero, representing the beginning of file. If ***nOrigin*** is the end of file, ***nOffset*** must be zero or negative.
nOrigin == 0	Seek from beginning of file
nOrigin == 1	Seek from the current pointer position
nOrigin == 2	Seek from end of file

FSEEK() returns the new position of the file pointer relative to the beginning of file (position 0) as an integer numeric value. This value is without regard to the original position of the file pointer. FSEEK() is a low-level file function that moves the file pointer forward or backward in an open binary file without actually reading the contents of the specified file. The beginning position and offset are specified as function arguments, and the new file position is returned.

FWRITE()

File write

`FWRITE(`***nHandle***`, `***cBuffer***`, [`***nBytes***`]) ⇒ `***nBytesWritten***

nHandle	is the file handle obtained from FOPEN(), FCREATE(), or predefined by DOS.
cBuffer	is the character string to write to the specified file.
nBytes	indicates the number of bytes to write beginning at the current file pointer position. If omitted, the entire content of ***cBuffer*** is written.

FWRITE() returns the number of bytes written as an integer numeric value. If the value returned is equal to ***nBytes***, the operation was successful. If the return value is less than ***nBytes*** or zero, either the disk is full or another error has occurred.

GETENV()

Get environment

`GETENV(`***cEnvironmentVariable***`) ⇒ `***cString***

cEnvironmentVariable	is the name of the DOS environment variable. When specifying this argument, you can use any combination of upper and lowercase letters; GETENV() is not case- sensitive.

GETENV() returns the contents of the specified DOS environment variable as a character string. If the variable cannot be found, GETENV() returns a null string ("").

HARDCR()

Hard carriage return

`HARDCR(`***cString***`) ⇒ `***cConvertedString***

cString	is the character string or memo field to convert.

HARDCR() is a memo function that replaces all soft carriage returns (CHR(141)) with hard carriage returns (CHR(13)). It is used to display long character strings and memo fields containing soft carriage returns with console commands.

HEADER()

`HEADER() ⇒ `***nBytes***

HEADER() returns the number of bytes in the header of the current database file as an integer numeric value. If no database file is in use, HEADER() returns a zero (0).

I2BIN()

Integer to binary

`I2BIN(`***nInteger***`) ⇒ `***cBinaryInteger***

nInteger	is an integer numeric value to convert. Decimal digits are truncated.

I2BIN() returns a two-byte character string containing a 16-bit binary integer.

IF()

«

```
[I]IF(lCondition, expTrue, expFalse) ⇒ Value
```

lCondition	is a logical expression to be evaluated.
expTrue	is the value, a condition-expression, of any data type, returned if ***lCondition*** is true ('`.T.`').
expFalse	is the value, of any date type, returned if ***lCondition*** is false ('`.F.`'). This argument need not be the same data type as ***expTrue***.

IF() returns the evaluation of ***expTrue*** if ***lCondition*** evaluates to true ('`.T.`') and ***expFalse*** if it evaluates to false ('`.F.`').

INDEXEXT()

«

Index extention

```
INDEXEXT() ⇒ cExtension
```

INDEXEXT() returns the default index file extension by determining which database driver is currently linked.

INDEXKEY()

«

```
INDEXKEY(nOrder) ⇒ cKeyExp
```

nOrder	is the ordinal position of the index in the list of index files opened by the last USE...INDEX or SET INDEX TO command for the current work area. A zero value specifies the controlling index, without regard to its actual position in the list.

INDEXKEY() returns the key expression of the specified index as a character string. If there is no corresponding index or if no database file is open, INDEXKEY() returns a null string ("").

INDEXORD()

«

Index order

```
INDEXORD() ⇒ nOrder
```

INDEXORD() returns an integer numeric value. The value returned is equal to the position of the controlling index in the list of open indexes for the current work area. A value of zero indicates that there is no controlling index and records are being accessed in natural order. If no database file is open, INDEXORD() will also return a zero.

INKEY()

«

Input key

```
INKEY([nSeconds]) ⇒ nInkeyCode
```

nSeconds	specifies the number of seconds INKEY() waits for a keypress. You can specify the value in increments as small as one-tenth of a second. Specifying zero halts the program until a key is pressed. If ***nSeconds*** is omitted, INKEY() does not wait for a keypress.

INKEY() returns an integer numeric value from -39 to 386, identifying the key extracted from the keyboard buffer. If the keyboard buffer is empty, INKEY() returns zero. INKEY() returns values for all ASCII characters, function, Alt+function, Ctrl+function, Alt+letter, and Ctrl+letter key combinations.

nInkeyCode value	Key or key combination
5	[*Up arrow*], [*Ctrl*]+[*E*]
24	[*Down arrow*], [*Ctrl*]+[*X*]
19	[*Left arrow*], [*Ctrl*]+[*S*]
4	[*Right arrow*], [*Ctrl*]+[*D*]
1	[*Home*], [*Ctrl*]+[*A*]
6	[*End*], [*Ctrl*]+[*F*]
18	[*PgUp*], [*Ctrl*]+[*R*]
3	[*PgDn*], [*Ctrl*]+[*C*]
397	[*Ctrl*]+[*Up arrow*]
401	[*Ctrl*]+[*Down arrow*]
26	[*Ctrl*]+[*Left arrow*], [*Ctrl*]+[*Z*]
2	[*Ctrl*]+[*Right arrow*], [*Ctrl*]+[*B*]
29	[*Ctrl*]+[*Home*]
23	[*Ctrl*]+[*End*], [*Ctrl*]+[*W*]
31	[*Ctrl*]+[*PgUp*], [*Ctrl*]+[*Hyphen*]
30	[*Ctrl*]+[*PgDn*], [*Ctrl*]+[^]
408	[*Alt*]+[*Up arrow*]
416	[*Alt*]+[*Down arrow*]
411	[*Alt*]+[*Left arrow*]
413	[*Alt*]+[*Right arrow*]
407	[*Alt*]+[*Home*]
415	[*Alt*]+[*End*]
409	[*Alt*]+[*PgUp*]
417	[*Alt*]+[*PgDn*]
13	[*Enter*], [*Ctrl*]+[*M*]
32	[*Space bar*]
27	[*Esc*]
10	[*Ctrl*]+[*Enter*]
379	[*Ctrl*]+[*Print Screen*]
309	[*Ctrl*]+[*?*]
284	[*Alt*]+[*Enter*]
387	[*Alt*]+[*Equals*]
257	[*Alt*]+[*Esc*]
422	Keypad [*Alt*]+[*Enter*]
399	Keypad [*Ctrl*]+[*5*]
405	Keypad [*Ctrl*]+[*/*]
406	Keypad [*Ctrl*]+[***]
398	Keypad [*Ctrl*]+[*-*]
400	Keypad [*Ctrl*]+[*+*]
5	Keypad [*Alt*]+[*5*]
420	Keypad [*Alt*]+[*/*]
311	Keypad [*Alt*]+[***]
330	Keypad [*Alt*]+[*-*]
334	Keypad [*Alt*]+[*+*]
22	[*Ins*], [*Ctrl*]+[*V*]
7	[*Del*], [*Ctrl*]+[*G*]
8	[*Backspace*], [*Ctrl*]+[*H*]
9	[*Tab*], [*Ctrl*]+[*I*]
271	[*Shift*]+[*Tab*]
402	[*Ctrl*]+[*Ins*]
403	[*Ctrl*]+[*Del*]
127	[*Ctrl*]+[*Backspace*]
404	[*Ctrl*]+[*Tab*]
418	[*Alt*]+[*Ins*]
419	[*Alt*]+[*Del*]
270	[*Alt*]+[*Backspace*]
421	[*Alt*]+[*Tab*]
1	[*Ctrl*]+[*A*], [*Home*]
2	[*Ctrl*]+[*B*], [*Ctrl*]+[*Right arrow*]
3	[*Ctrl*]+[*C*], [*PgDn*], [*Ctrl*]+[*ScrollLock*]
4	[*Ctrl*]+[*D*], [*Right arrow*]
5	[*Ctrl*]+[*E*], [*Up arrow*]
6	[*Ctrl*]+[*F*], [*End*]
7	[*Ctrl*]+[*G*], [*Del*]
8	[*Ctrl*]+[*H*], [*Backspace*]
9	[*Ctrl*]+[*I*], [*Tab*]

nInkeyCode value	Key or key combination
10	[*Ctrl*]+[*J*]
11	[*Ctrl*]+[*K*]
12	[*Ctrl*]+[*L*]
13	[*Ctrl*]+[*M*], [*Return*]
14	[*Ctrl*]+[*N*]
15	[*Ctrl*]+[*O*]
16	[*Ctrl*]+[*P*]
17	[*Ctrl*]+[*Q*]
18	[*Ctrl*]+[*R*], [*PgUp*]
19	[*Ctrl*]+[*S*], [*Left arrow*]
20	[*Ctrl*]+[*T*]
21	[*Ctrl*]+[*U*]
22	[*Ctrl*]+[*V*], [*Ins*]
23	[*Ctrl*]+[*W*], [*Ctrl*]+[*End*]
24	[*Ctrl*]+[*X*], [*Down arrow*]
25	[*Ctrl*]+[*Y*]
26	[*Ctrl*]+[*Z*], [*Ctrl*]+[*Left arrow*]
286	[*Alt*]+[*A*]
304	[*Alt*]+[*B*]
302	[*Alt*]+[*C*]
288	[*Alt*]+[*D*]
274	[*Alt*]+[*E*]
289	[*Alt*]+[*F*]
290	[*Alt*]+[*G*]
291	[*Alt*]+[*H*]
279	[*Alt*]+[*I*]
292	[*Alt*]+[*J*]
293	[*Alt*]+[*K*]
294	[*Alt*]+[*L*]
306	[*Alt*]+[*M*]
305	[*Alt*]+[*N*]
280	[*Alt*]+[*O*]
281	[*Alt*]+[*P*]
272	[*Alt*]+[*Q*]
275	[*Alt*]+[*R*]
287	[*Alt*]+[*S*]
276	[*Alt*]+[*T*]
278	[*Alt*]+[*U*]
303	[*Alt*]+[*V*]
273	[*Alt*]+[*W*]
301	[*Alt*]+[*X*]
277	[*Alt*]+[*Y*]
300	[*Alt*]+[*Z*]
376	[*Alt*]+[*1*]
377	[*Alt*]+[*2*]
378	[*Alt*]+[*3*]
379	[*Alt*]+[*4*]
380	[*Alt*]+[*5*]
381	[*Alt*]+[*6*]
382	[*Alt*]+[*7*]
383	[*Alt*]+[*8*]
384	[*Alt*]+[*9*]
385	[*Alt*]+[*0*]
28	[*F1*], [*Ctrl*]+[*Backslash*]
-1	[*F2*]
-2	[*F3*]
-3	[*F4*]
-4	[*F5*]
-5	[*F6*]
-6	[*F7*]
-7	[*F8*]
-8	[*F9*]
-9	[*F10*]
-40	[*F11*]
-41	[*F12*]
-20	[*Ctrl*]+[*F1*]
-21	[*Ctrl*]+[*F2*]
-22	[*Ctrl*]+[*F4*]
-23	[*Ctrl*]+[*F3*]
-24	[*Ctrl*]+[*F5*]
-25	[*Ctrl*]+[*F6*]

nInkeyCode value	Key or key combination
-26	[*Ctrl*]+[*F7*]
-27	[*Ctrl*]+[*F8*]
-28	[*Ctrl*]+[*F9*]
-29	[*Ctrl*]+[*F10*]
-44	[*Ctrl*]+[*F11*]
-45	[*Ctrl*]+[*F12*]
-30	[*Alt*]+[*F1*]
-31	[*Alt*]+[*F2*]
-32	[*Alt*]+[*F3*]
-33	[*Alt*]+[*F4*]
-34	[*Alt*]+[*F5*]
-35	[*Alt*]+[*F6*]
-36	[*Alt*]+[*F7*]
-37	[*Alt*]+[*F8*]
-38	[*Alt*]+[*F9*]
-39	[*Alt*]+[*F10*]
-46	[*Alt*]+[*F11*]
-47	[*Alt*]+[*F12*]
-10	[*Shift*]+[*F1*]
-11	[*Shift*]+[*F2*]
-12	[*Shift*]+[*F3*]
-13	[*Shift*]+[*F4*]
-14	[*Shift*]+[*F5*]
-15	[*Shift*]+[*F6*]
-16	[*Shift*]+[*F7*]
-17	[*Shift*]+[*F8*]
-18	[*Shift*]+[*F9*]
-19	[*Shift*]+[*F10*]
-42	[*Shift*]+[*F11*]
-43	[*Shift*]+[*F12*]

INT()

«

Integer

```
INT(nExp) ⇒ nInteger
```

nExp	is a numeric expression to convert to an integer.

INT() is a numeric function that converts a numeric value to an integer by truncating all digits to the right of the decimal point. INT() is useful in operations where the decimal portion of a number is not needed.

ISALPHA()

«

```
ISALPHA(cString) ⇒ lBoolean
```

cString	is the character string to examine.

ISALPHA() returns true (‘**.T.**’) if the first character in ***cString*** is alphabetic; otherwise, it returns false (‘**.F.**’).

ISCOLOR()

«

```
ISCOLOR() | ISCOLOUR() ⇒ lBoolean
```

ISCOLOR() returns true (‘**.T.**’) if there is a color graphics card installed; otherwise, it returns false (‘**.F.**’).

ISDIGIT()

«

```
ISDIGIT(cString) ⇒ lBoolean
```

cString	is the character string to examine.

ISDIGIT() returns true (‘**.T.**’) if the first character of the character string is a digit between zero and nine; otherwise, it returns false

(‘**.F.**’).

ISLOWER()

«

```
ISLOWER(cString) ⇒ lBoolean
```

cString	is the character string to examine.

ISLOWER() returns true (‘**.T.**’) if the first character of the character string is a lowercase letter; otherwise, it returns false (‘**.F.**’).

ISPRINTER()

«

```
ISPRINTER() ⇒ lReady
```

ISPRINTER() returns true (‘**.T.**’) if ‘LPT1:’ is ready; otherwise, it returns false (‘**.F.**’).

ISUPPER()

«

```
ISUPPER(cString) ⇒ lBoolean
```

cString	is the character string to examine.

ISUPPER() returns true (‘**.T.**’) if the first character is an uppercase letter; otherwise, it returns false (‘**.F.**’).

L2BIN()

«

Long to binary

```
L2BIN(nExp) ⇒ cBinaryInteger
```

nExp	is the numeric value to convert. Decimal digits are truncated.

L2BIN() returns a four-byte character string formatted as a 32- bit binary integer.

LASTKEY()

«

```
LASTKEY() ⇒ nInkeyCode
```

LASTKEY() is a keyboard function that reports the INKEY() value of the last key fetched from the keyboard buffer by the INKEY() function, or a wait state. LASTKEY() retains its current value until another key is fetched from the keyboard buffer.

LASTREC()

«

Last record

```
LASTREC() ⇒ nRecords
```

LASTREC() returns the number of physical records in the active alias as an integer numeric value.

LEFT()

«

```
LEFT(cString, nCount) ⇒ cSubString
```

cString	is a character string from which to extract characters.
nCount	is the number of characters to extract.

LEFT() returns the leftmost ***nCount*** characters of ***cString*** as a character string. If ***nCount*** is negative or zero, LEFT() returns a null string (""). If ***nCount*** is larger than the length of the character string, LEFT() returns the entire string.

LEN()

«

Length

```
LEN(cString | aTarget) ⇒ nCount
```

cString	is the character string to count.
aTarget	is the array to count.

LEN() returns the length of a character string or the number of elements in an array as an integer numeric value.

LOG()

«

```
LOG(nExp) ⇒ nNaturalLog
```

nExp	is a numeric value greater than zero to convert to its natural logarithm.

LOG() returns the natural logarithm as a numeric value. If ***nExp*** is less than or equal to zero, LOG() returns a numeric overflow (displayed as a row of asterisks).

LOWER()

«

```
LOWER(cString) ⇒ cLowerString
```

cString	is a character string to convert to lowercase.

LOWER() returns a copy of ***cString*** with all alphabetic characters converted to lowercase.

LTRIM()

«

Left trim

```
LTRIM(cString) ⇒ cTrimString
```

cString	is the character string to copy without leading spaces.

LTRIM() returns a copy of ***cString*** with the leading spaces removed.

LUPDATE()

«

Last update

```
LUPDATE() ⇒ dModification
```

LUPDATE() returns the date of last change to the open database file in the current work area.

MAX()

«

```
MAX(nExp1, nExp2) ⇒ nLarger
```

```
MAX(dExp1, dExp2) ⇒ dLarger
```

nExp1, ***nExp2***	are the numeric values to compare.
dExp1, ***dExp2***	are the date values to compare.

MAX() returns the larger of the two arguments. The value returned is the same type as the arguments.

MAXCOL()

«

Max column

```
MAXCOL() ⇒ nColumn
```

MAXCOL() returns the column number of the rightmost visible column for display purposes.

MAXROW()

```
MAXROW() ⇒ nRow
```

MAXROW() returns the row number of the bottommost visible row for display purposes.

MEMOEDIT()

```
MEMOEDIT( [cString],
      [nTop], [nLeft],
      [nBottom], [nRight],
      [lEditMode],
      [cUserFunction],
      [nLineLength],
      [nTabSize],
      [nTextBufferRow],
      [nTextBufferColumn],
      [nWindowRow],
      [nWindowColumn] ) ⇒ cTextBuffer
```

cString	is the character string or memo field to copy to the MEMOEDIT() text buffer.
nTop, ***nLeft***, ***nBottom***, ***nRight***	are window coordinates. The default coordinates are 0, 0, MAXROW(), and MAXCOL().
lEditMode	determines whether the text buffer can be edited or merely displayed. If not specified, the default value is true (‘**.T.**’).
cUserFunction	is the name of a user-defined function that executes when the user presses a key not recognized by MEMOEDIT() and when no keys are pending in the keyboard buffer.
nLineLength	determines the length of lines displayed in the MEMOEDIT() window. If a line is greater than ***nLineLength***, it is word wrapped to the next line in the MEMOEDIT() window. The default line length is (***nRight*** - ***nLeft***).
nTabSize	determines the size of a tab character to insert when the user presses Tab. The default is four.
nTextBufferRow, ***nTextBufferColumn***	define the display position of the cursor within the text buffer when MEMOEDIT() is invoked. ***nTextBufferRow*** begins with one and ***nTextBufferColumn*** begins with zero. Default is the beginning of MEMOEDIT() window.
nWindowRow, ***nWindowColumn***	define the initial position of the cursor within the MEMOEDIT() window. Row and column positions begin with zero. If these arguments are not specified, the initial window position is row zero and the current cursor column position.

MEMOEDIT() is a user interface and general purpose text editing function that edits memo fields and long character strings. Editing occurs within a specified window region placed anywhere on the screen.

[*Uparrow*]/[*Ctrl*]+E	Move up one line
[*Dnarrow*]/[*Ctrl*]+X	Move down one line
[*Leftarrow*]/[*Ctrl*]+S	Move left one character
[*Rightarrow*]/[*Ctrl*]+D	Move right one character
[*Ctrl*]-[*Leftarrow*]/[*Ctrl*]+A	Move left one word
[*Ctrl*]-[*Rightarrow*]/[*Ctrl*]+F	Move right one word
[*Home*]	Move to beginning of current line
[*End*]	Move to end of current line
[*Ctrl*]+[*Home*]	Move to beginning of current window
[*Ctrl*]+[*End*]	Move to end of current window
[*PgUp*]	Move to previous edit window
[*PgDn*]	Move to next edit window
[*Ctrl*]+[*PgUp*]	Move to beginning of memo
[*Ctrl*]+[*PgDn*]	Move to end of memo
[*Return*]	Move to beginning of next line
[*Delete*]	Delete character at cursor
[*Backspace*]	Delete character to left of cursor
[*Tab*]	Insert tab character or spaces
Printable characters	Insert character
[*Ctrl*]+Y	Delete the current line
[*Ctrl*]+T	Delete word right
[*Ctrl*]+B	Reform paragraph
[*Ctrl*]+V/[*Ins*]	Toggle insert mode
[*Ctrl*]+W	Finish editing with save
[*Esc*]	Abort edit and return original

MEMOLINE()

```
MEMOLINE(cString,
      [nLineLength],
      [nLineNumber],
      [nTabSize],
      [lWrap] ) ⇒ cLine
```

cString	is the memo field or character string from which to extract a line of text.
nLineLength	specifies the number of characters per line and can be between four and 254 . If not specified, the default line length is 79.
nLineNumber	is the line number to extract. If not specified, the default value is one.
nTabSize	defines the tab size. If not specified, the default value is four.
lWrap	toggles word wrap on and off. Specifying true (‘**.T.**’) toggles word wrap on; false (‘**.F.**’) toggles it off. If not specified, the default value is true (‘**.T.**’).

MEMOLINE() returns the line of text specified by ***nLineNumber*** in ***cString*** as a character string. If the line has fewer characters than the indicated length, the return value is padded with blanks. If the line number is greater than the total number of lines in ***cString***, MEMOLINE() returns a null string (""). If ***lWrap*** is true (‘**.T.**’) and the indicated line length breaks the line in the middle of a word, that word is not included as part of the return value but shows up at the beginning of the next line extracted with MEMOLINE(). If ***lWrap*** is false (‘**.F.**’), MEMOLINE() returns only the number of characters specified by the line length. The next line extracted by MEMOLINE() begins with the character following the next hard carriage return, and all intervening characters are not processed.

MEMOREAD()

```
MEMOREAD(cFile) ⇒ cString
```

cFile	is the name of the file to read from disk. It must include an extension if there is one, and can optionally include a path.

MEMOREAD() returns the contents of a text file as a character string.

MEMORY()

```
MEMORY(nExp) ⇒ nKbytes
```

nExp	is a numeric value that determines the type of value MEMORY() returns.

MEMORY() returns an integer numeric value representing the amount of memory available.

MEMORY(0)	Estimated total space available for character values
MEMORY(1)	Largest contiguous block available for character values
MEMORY(2)	Area available for RUN commands

MEMOTRAN()

Memo translate

```
MEMOTRAN(cString,
      [cReplaceHardCR],
      [cReplaceSoftCR]) ⇒ cNewString
```

cString	is the character string or memo field to search.
cReplaceHardCR	is the character to replace a hard carriage return/linefeed pair with. If not specified, the default value is a semicolon (;).
cReplaceSoftCR	is the character to replace a soft carriage return/linefeed pair with. If not specified, the default value is a space.

MEMOTRAN() returns a copy of ***cString*** with the specified carriage return/linefeed pairs replaced.

MEMOWRIT()

Memo write

```
MEMOWRIT(cFile, cString) ⇒ lSuccess
```

cFile	is the name of the target disk file including the file extension and optional path and drive designator.
cString	is the character string or memo field to write to ***cFile***.

MEMOWRIT() is a memo function that writes a character string or memo field to a disk file. If a path is not specified, MEMOWRIT() writes ***cFile*** to the current DOS directory and not the current DEFAULT directory. If ***cFile*** already exists, it is overwritten. MEMOWRIT() returns true (‘**.T.**’) if the writing operation is successful; otherwise, it returns false (‘**.F.**’).

MEMVARBLOCK()

```
MEMVARBLOCK(cMemvarName) ⇒ bMemvarBlock
```

cMemvarName	is the name of the variable referred to by the set-get block, specified as a character string.

MEMVARBLOCK() returns a code block that when evaluated sets (assigns) or gets (retrieves) the value of the given memory variable. If ***cMemvarName*** does not exist, MEMVARBLOCK() returns NIL.

MIN()

```
MIN(nExp1, nExp2) ⇒ nSmaller
```

```
MIN(dExp1, dExp2) ⇒ dSmaller
```

nExp1, ***nExp2***	are the numeric values to compare.
dExp1, ***dExp2***	are the date values to compare.

MIN() returns the smaller of the two arguments. The value returned is the same data type as the arguments.

MLCOUNT()

Memo line count

```
MLCOUNT(cString, [nLineLength],
      [nTabSize], [lWrap]) ⇒ nLines
```

cString	is the character string or memo field to count.
nLineLength	specifies the number of characters per line and can range from four to 254 . If not specified, the default line length is 79.
nTabSize	defines the tab size. If not specified, the default value is four.
lWrap	toggles word wrap on and off. Specifying true (‘**.T.**’) toggles word wrap on; false (‘**.F.**’) toggles it off. If not specified, the default value is true (‘**.T.**’).

MLCOUNT() returns the number of lines in ***cString*** depending on the ***nLineLength***, the ***nTabSize***, and whether word wrapping is on or off.

MLCTOPOS()

Memo line column to position

```
MLCTOPOS(cText, nWidth, nLine,
      nCol, [nTabSize], [lWrap]) ⇒ nPosition
```

cText	is the text string to scan.
nWidth	is the line length formatting width.
nLine	is the line number counting from 1.
nCol	is the column number counting from 0.
nTabSize	is the number of columns between tab stops. If not specified, the default is 4.
lWrap	is the word wrap flag. If not specified, the default is true (‘**.T.**’).

MLCTOPOS() returns the byte position within ***cText*** counting from 1.

MLPOS()

Memo line position

```
MLPOS(cString, nLineLength,
      nLine, [nTabSize], [lWrap]) ⇒ nPosition
```

cString	is a character string or memo field.
nLineLength	specifies the number of characters per line.
nLine	specifies the line number.
nTabSize	defines the tab size. The default is four.
lWrap	toggles word wrap on and off. Specifying true (‘**.T.**’) toggles word wrap on, and false (‘**.F.**’) toggles it off. The default is true (‘**.T.**’).

MLPOS() returns the character position of ***nLine*** in ***cString*** as an integer numeric value. If ***nLine*** is greater than the number of lines in ***cString***, MLPOS() returns the length of ***cString***.

MONTH()

```
MONTH(dDate) ⇒ nMonth
```

dDate	is the date value to convert.

MONTH() returns an integer numeric value in the range of zero to 12. Specifying a null date (CTOD("")) returns zero.

MPOSTOLC()

Memo position to line column

```
MPOSTOLC(cText, nWidth, nPos,
      [nTabSize], [lWrap]) ⇒ aLineColumn
```

cText	is a text string.
nWidth	is the length of the formatted line.
nPos	is the byte position within text counting from one.
nTabSize	is the number of columns between tab stops. If not specified, the default is four.
lWrap	is the word wrap flag. If not specified, the default is true (‘**.T.**’).

MPOSTOLC() returns an array containing the line and the column values for the specified byte position, ***nPos***. MPOSTOLC() is a memo function that determines the formatted line and column corresponding to a particular byte position within ***cText***. Note that the line number returned is one-relative, the column number is zero-relative. This is compatible with MEMOEDIT(). ***nPos*** is one-relative, compatible with AT(), RAT(), and other string functions.

NETERR()

Net error

```
NETERR([lNewError]) ⇒ lError
```

lNewError	if specified sets the value returned by NETERR() to the specified status. ***lNewError*** can be either true (‘**.T.**’) or false (‘**.F.**’). Setting NETERR() to a specified value allows the runtime error handler to control the way certain file errors are handled.

NETERR() returns true (‘**.T.**’) if a USE or APPEND BLANK fails. The initial value of NETERR() is false (‘**.F.**’). If the current process is not running under a network operating system, NETERR() always returns false (‘**.F.**’).

NETNAME()

```
NETNAME() ⇒ cWorkstationName
```

NETNAME() returns the workstation identification as a character string up to 15 characters in length. If the workstation identification was never set or the application is not operating under the IBM PC Network, it returns a null string ("").

NEXTKEY()

```
NEXTKEY() ⇒ nInkeyCode
```

NEXTKEY() returns an integer numeric value ranging from -39 to 386. If the keyboard buffer is empty, NEXTKEY() returns zero. If SET TYPEAHEAD is zero, NEXTKEY() always returns zero. NEXTKEY() is like the INKEY() function, but differs in one fundamental respect. INKEY() removes the pending key from the keyboard buffer and updates LASTKEY() with the value of the key. NEXTKEY(), by contrast, reads, but does not remove the key from the keyboard buffer and does not update LASTKEY().

NOSNOW()

```
NOSNOW(lToggle) ⇒ NIL
```

lToggle	is a logical value that toggles the current state of snow suppression. A value of true (‘**.T.**’) enables the snow suppression on, while a value of false (‘**.F.**’) disables snow suppression.

NOSNOW() is used to suppress snow on old CGA monitors.

ORDBAGEXT()

```
ORDBAGEXT() ⇒ cBagExt
```

ORDBAGEXT() returns a character expression that is the default Order Bag extension of the current work area. cBagExt is determined by the RDD active in the current work area.

ORDBAGNAME()

```
ORDBAGNAME(nOrder | cOrderName) ⇒ cOrderBagName
```

nOrder	is an integer that identifies the position in the Order List of the target Order whose Order Bag name is sought.
cOrderName	is a character string that represents the name of the target Order whose Order Bag name is sought.

ORDBAGNAME() returns a character string, the Order Bag name of the specific Order.

ORDCREATE()

```
ORDCREATE(cOrderBagName, [cOrderName], cExpKey, [bExpKey],
      [lUnique]) ⇒ NIL
```

cOrderBagName	is the name of a disk file containing one or more Orders.
cOrderName	is the name of the Order to be created.
cExpKey	is an expression that returns the key value to place in the Order for each record in the current work area. The maximum length of the index key expression is determined by the database driver.
bExpKey	is a code block that evaluates to a key value that is placed in the Order for each record in the current work area.
lUnique	specifies whether a unique Order is to be created. Default is the current global _SET_UNIQUE setting.

ORDCREATE() is an Order management function that creates an Order in the current work area. It works like DBCREATEINDEX() except that it lets you create Orders in RDDs that recognize multiple Order Bags.

ORDDESTROY()

```
ORDDESTROY(cOrderName [, cOrderBagName ]) ⇒ NIL
```

cOrderName	is the name of the Order to be removed from the current or specified work area.
cOrderBagName	is the name of a disk file containing one or more Orders.

ORDDESTROY() is an Order management function that removes a specified Order from multiple-Order Bags. ORDDESTROY() is not supported for DBFNDX and DBFNTX.

ORDFOR()

```
ORDFOR(cOrderName | nOrder [, cOrderBagName]) ⇒ cForExp
```

cOrderName	is the name of the target Order, whose cForExp is sought.
nOrder	is an integer that identifies the position in the Order List of the target Order whose cForExp is sought.
cOrderBagName	is the name of an Order Bag containing one or more Orders.

ORDFOR() returns a character expression, cForExp, that represents the FOR condition of the specified Order. If the Order was not created using the FOR clause the return value will be an empty string (""). If the database driver does not support the FOR condition, it may either return an empty string ("") or raise an "unsupported function" error, depending on the driver.

ORDKEY()

```
ORDKEY(cOrderName | nOrder [, cOrderBagName]) ⇒ cExpKey
```

cOrderName	is the name of an Order, a logical ordering of a database.
nOrder	is an integer that identifies the position in the Order List of the target Order whose cExpKey is sought.
cOrderBagName	is the name of a disk file containing one or more Orders.

ORDKEY() is an Order management function that returns a character expression, cExpKey, that represents the key expression of the specified Order.

ORDLISTADD()

```
ORDLISTADD(cOrderBagName [, cOrderName]) ⇒ NIL
```

cOrderBagName	is the name of a disk file containing one or more Orders.
cOrderName	the name of the specific Order from the Order Bag to be added to the Order List of the current work area. If you do not specify ***cOrderName***, all orders in the Order Bag are added to the Order List of the current work area.

ORDLISTADD() is an Order management function that adds the contents of an Order Bag , or a single Order in an Order Bag, to the Order List. Any Orders already associated with the work area continue to be active. If the newly opened Order Bag contains the only Order associated with the work area, it becomes the controlling Order; otherwise, the controlling Order remains unchanged.

ORDLISTCLEAR()

```
ORDLISTCLEAR() ⇒ NIL
```

ORDLISTCLEAR() is an Order management function that removes all Orders from the Order List for the current work area.

ORDLISTREBUILD()

```
ORDLISTREBUILD() ⇒ NIL
```

ORDLISTREBUILD() is an Order management function that rebuilds all the orders in the current Order List.

ORDNAME()

```
ORDNAME(nOrder[, cOrderBagName]) ⇒ cOrderName
```

nOrder	is an integer that identifies the position in the Order List of the target Order whose database name is sought.
cOrderBagName	is the name of a disk file containing one or more Orders.

ORDNAME() returns the name of the specified Order in the current Order List or the specified Order Bag if opened in the Current Order list.

ORDNUMBER()

```
ORDNUMBER(cOrderName [, cOrderBagName]) ⇒ nOrderNo
```

cOrderName	the name of the specific Order whose position in the Order List is sought.
cOrderBagName	is the name of a disk file containing one or more Orders.

ORDNUMBER() returns nOrderNo, an integer that represents the position of the specified Order in the Order List.

ORDSETFOCUS()

```
ORDSETFOCUS([cOrderName | nOrder] [, cOrderBagName])
      ⇒ cPrevOrderNameInFocus
```

cOrderName	is the name of the selected Order, a logical ordering of a database.
nOrder	is a number representing the position in the Order List of the selected Order.
cOrderBagName	is the name of a disk file containing one or more Orders.

ORDSETFOCUS() is an Order management function that returns the Order Name of the previous controlling Order and optionally sets the focus to an new Order.

OS()

```
OS() ⇒ cOsName
```

OS() returns the operating system name as a character string.

OUTERR()

Output error

```
OUTERR(exp_list) ⇒ NIL
```

exp_list	is a list of values to display and can consist of any combination of data types including memo.

OUTERR() is identical to OUTSTD() except that it writes to the standard error device rather than the standard output device. Output sent to the standard error device bypasses the console and output devices as well as any DOS redirection. It is typically used to log error messages in a manner that will not interfere with the standard screen or printer output.

OUTSTD()

Output standard

```
OUTSTD(exp_list) ⇒ NIL
```

exp_list	is a list of values to display and can consist of any combination of data types including memo.

OUTSTD() is a simple output function similar to QOUT(), except that it writes to the STDOUT device (instead of to the console output stream).

PAD?()

«

```
PADL(exp, nLength, [cFillChar]) ⇒ cPaddedString
```

```
PADC(exp, nLength, [cFillChar]) ⇒ cPaddedString
```

```
PADR(exp, nLength, [cFillChar]) ⇒ cPaddedString
```

exp	is a character, numeric, or date value to pad with a fill character.
nLength	is the length of the character string to return.
cFillChar	is the character to pad ***exp*** with. If not specified, the default is a space character.

PADC(), PADL(), and PADR() are character functions that pad character, date, and numeric values with a fill character to create a new character string of a specified length. PADC() centers ***exp*** within ***nLength*** adding fill characters to the left and right sides; PADL() adds fill characters on the left side; and PADR() adds fill characters on the right side.

PCOL()

«

Printed column

```
PCOL() ⇒ nColumn
```

PCOL() returns an integer numeric value representing the last printed column position, plus one. The beginning column position is zero.

PROW()

«

Printed row

```
PROW() ⇒ nRow
```

PROW() returns an integer numeric value that represents the number of the current line sent to the printer. The beginning row position is zero.

QOUT()

«

```
QOUT([exp_list]) ⇒ NIL
```

```
QQOUT([exp_list]) ⇒ NIL
```

exp_list	is a comma-separated list of expressions (of any data type other than array or block) to display to the console. If no argument is specified and QOUT() is specified, a carriage return/linefeed pair is displayed. If QQOUT() is specified without arguments, nothing displays.

QOUT() and QQOUT() are console functions. They display the results of one or more expressions to the console. QOUT() outputs carriage return and linefeed characters before displaying the results of ***exp_list***. QQOUT() displays the results of ***exp_list*** at the current ROW() and COL() position. When QOUT() and QQOUT() display

to the console, ROW() and COL() are updated.

RAT()

«

Right at

```
RAT(cSearch, cTarget) ⇒ nPosition
```

cSearch	is the character string to locate.
cTarget	is the character string to search.

RAT() returns the position of ***cSearch*** within ***cTarget*** as an integer numeric value, starting the search from the right. If ***cSearch*** is not found, RAT() returns zero.

RDDLIST()

«

```
RDDLIST([nRDDType]) ⇒ aRDDList
```

nRDDType	is an integer that represents the type of the RDD you wish to list. ***nRDDType*** = 1 Full RDD implementation ***nRDDType*** = 2 Import/Export only driver.

RDDLIST() returns a one-dimensional array of the RDD names registered with the application as ***nRDDType***.

RDDNAME()

«

```
RDDNAME() ⇒ cRDDName
```

RDDNAME() returns a character string, cRDDName, the registered name of the active RDD in the current or specified work area.

RDDSETDEFAULT()

«

```
RDDSETDEFAULT([cNewDefaultRDD]) ⇒ cPreviousDefaultRDD
```

cNewDefaultRDD	is a character string, the name of the RDD that is to be made the new default RDD in the application.

RDDSETDEFAULT() is an RDD function that sets or returns the name of the previous default RDD driver and, optionally, sets the current driver to the new RDD driver specified by ***cNewDefaultRDD***.

READINSERT()

«

```
READINSERT([lToggle]) ⇒ lCurrentMode
```

lToggle	toggles the insert mode on or off. True (‘**.T.**’) turns insert on, while false (‘**.F.**’) turns insert off. The default is false (‘**.F.**’) or the last user-selected mode in READ or MEMOEDIT().

READINSERT() returns the current insert mode state as a logical value.

READMODAL()

«

```
READMODAL(aGetList) ⇒ NIL
```

aGetList	is an array containing a list of Get objects to edit.

READMODAL() is like the READ command, but takes a GetList array as an argument and does not reinitialize the GetList array when it terminates. The GET system is implemented using a public ar-

ray called GetList. Each time an @...GET command executes, it creates a Get object and adds to the currently visible GetList array. The standard READ command is preprocessed into a call to READMODAL() using the GetList array as its argument.

READVAR()

```
READVAR() ⇒ cVarName
```

READVAR() returns the name of the variable associated with the current Get object or the variable being assigned by the current MENU TO command as an uppercase character string.

RECNO()

Record number

```
RECNO() ⇒ nRecord
```

RECNO() returns the current record number as an integer numeric value. If the work area contains a database file with zero records, RECNO() returns one, BOF() and EOF() both return true (‘`.T.`’), and LASTREC() returns zero. If the record pointer is moved past the last record, RECNO() returns LASTREC() + 1 and EOF() returns true (‘`.T.`’). If an attempt is made to move before the first record, RECNO() returns the record number of the first logical record in the database file and BOF() returns true (‘`.T.`’). If no database file is open, RECNO() will return a zero.

RECSIZE()

Record size

```
RECSIZE() ⇒ nBytes
```

RECSIZE() returns, as a numeric value, the record length, in bytes, of the database file open in the current work area. RECSIZE() returns zero if no database file is open.

REPLICATE()

```
REPLICATE(cString, nCount) ⇒ cRepeatedString
```

cString	is the character string to repeat.
nCount	is the number of times to repeat ***cString***.

REPLICATE() returns a character string. Specifying a zero as the ***nCount*** argument returns a null string ("").

RESTSCREEN()

Restore screen

```
RESTSCREEN([nTop], [nLeft],
    [nBottom], [nRight], cScreen) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom***, ***nRight***	define the coordinates of the screen information contained in ***cScreen***. If the ***cScreen*** was saved without coordinates to preserve the entire screen, no screen coordinates are necessary with RESTSCREEN().
cScreen	is a character string containing the saved screen region.

RESTSCREEN() is a screen function that redisplays a screen region saved with SAVESCREEN(). The target screen location may be the same as or different than the original location when the screen region was saved.

RIGHT()

```
RIGHT(cString, nCount) ⇒ cSubString
```

cString	is the character string from which to extract characters.
nCount	is the number of characters to extract.

RIGHT() returns the rightmost ***nCount*** characters of ***cString***. If ***nCount*** is zero, RIGHT() returns a null string (""). If ***nCount*** is negative or larger than the length of the character string, RIGHT() returns ***cString***.

RLOCK()

Record lock

```
RLOCK() ⇒ lSuccess
```

RLOCK() is a network function that locks the current record, preventing other users from updating the record until the lock is released. RLOCK() provides a shared lock, allowing other users read-only access to the locked record while allowing only the current user to modify it. A record lock remains until another record is locked, an UNLOCK is executed, the current database file is closed, or an FLOCK() is obtained on the current database file.

ROUND()

```
ROUND(nNumber, nDecimals) ⇒ nRounded
```

nNumber	is the numeric value to round.
nDecimals	defines the number of decimal places to retain. Specifying a negative ***nDecimals*** value rounds whole number digits.

ROUND() is a numeric function that rounds ***nNumber*** to the number of places specified by ***nDecimals***. Specifying a zero or negative value for ***nDecimals*** allows rounding of whole numbers. A negative ***nDecimals*** indicates the number of digits to the left of the decimal point to round. Digits between five to nine, inclusive, are rounded up. Digits below five are rounded down.

ROW()

```
ROW() ⇒ nRow
```

ROW() returns the cursor row position as an integer numeric value. The range of the return value is zero to MAXROW().

RTRIM()

Right trim

```
[R]TRIM(cString) ⇒ cTrimString
```

cString	is the character string to copy without trailing spaces.

RTRIM() returns a copy of ***cString*** with the trailing spaces removed. If ***cString*** is a null string ("") or all spaces, RTRIM() returns a null string ("").

SAVESCREEN()

```
SAVESCREEN([nTop], [nLeft],
    [nBottom], [nRight]) ⇒ cScreen
```

nTop, ***nLeft***, ***nBottom***, ***nRight***	define the coordinates of the screen region to save. Default is the entire screen.

SAVESCREEN() returns the specified screen region as a character string.

SCROLL()

```
SCROLL([nTop], [nLeft],
    [nBottom], [nRight], [nVert] [nHoriz]) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom***, ***nRight*** define the scroll region coordinates.

nVert	defines the number of rows to scroll, vertically. A positive value scrolls up the specified number of rows. A negative value scrolls down the specified number of rows. A value of zero disables vertical scrolling. If ***nVert*** is not specified, zero is assumed.
nHoriz	defines the number of rows to scroll horizontally. A positive value scrolls left the specified number of columns. A negative value scrolls right the specified number of columns. A value of zero disables horizontal scrolling. If ***nHoriz*** is not specified, zero is assumed. If you supply neither ***nVert*** or ***nHoriz*** parameters to SCROLL(), the area specified by the first four parameters will be blanked.

SCROLL() is a screen function that scrolls a screen region up or down a specified number of rows. When a screen scrolls up, the first line of the region is erased, all other lines are moved up, and a blank line is displayed in the current standard color on the bottom line of the specified region. If the region scrolls down, the operation is reversed. If the screen region is scrolled more than one line, this process is repeated.

SECONDS()

```
SECONDS() ⇒ nSeconds
```

SECONDS() returns the system time as a numeric value in the form seconds.hundredths. The numeric value returned is the number of seconds elapsed since midnight, and is based on a twenty-four hour clock in a range from zero to 86399.

SELECT()

```
SELECT([cAlias]) ⇒ nWorkArea
```

cAlias	is the target work area alias name.

SELECT() returns the work area of the specified alias as a integer numeric value.

SET()

```
SET(nSpecifier, [expNewSetting], [lOpenMode])
    ⇒ CurrentSetting
```

nSpecifier	is a numeric value that identifies the setting to be inspected or changed.
expNewSetting	is an optional argument that specifies a new value for the ***nSpecifier***. The type of ***expNewSetting*** depends on ***nSpecifier***.
lOpenMode	is a logical value that indicates whether or not files are opened for some settings. A value of false (‘`.F.`’) means the file should be truncated. A value of true (‘`.T.`’) means the file should be opened in append mode. In either case, if the file does not exist, it is created. If this argument is not specified, the default is append mode.

SET() returns the current value of the specified setting.

Inside nB, the function SET() is not so easy to use as inside the Clipper environment. This because nB cannot support manifest constants and a numeric specifier ***nSpecifier*** is not easy to manage. Instead of SET() you can use SETVERB().

SETBLINK()

```
SETBLINK([lToggle]) ⇒ lCurrentSetting
```

lToggle	changes the meaning of the asterisk (*) character when it is encountered in a SETCOLOR() string. Specifying true (‘`.T.`’) sets character blinking on and false (‘`.F.`’) sets background intensity. The default is true (‘`.T.`’).

SETBLINK() returns the current setting as a logical value.

SETCANCEL()

```
SETCANCEL([lToggle]) ⇒ lCurrentSetting
```

lToggle	changes the availability of Alt-C and Ctrl-Break as termination keys. Specifying true (‘`.T.`’) allows either of these keys to terminate an application and false (‘`.F.`’) disables both keys. The default is true (‘`.T.`’).

SETCANCEL() returns the current setting as a logical value.

SETCOLOR()

```
SETCOLOR([cColorString]) ⇒ cColorString
```

cColorString	is a character string containing a list of color attribute settings for subsequent screen painting.

SETCURSOR()

```
SETCURSOR([nCursorShape]) ⇒ nCurrentSetting
```

nCursorShape	is a number indicating the shape of the cursor. ***nCursorShape*** == 0 None ***nCursorShape*** == 1 Underline ***nCursorShape*** == 2 Lower half block ***nCursorShape*** == 3 Full block ***nCursorShape*** == 4 Upper half block

SETCURSOR() returns the current cursor shape as a numeric value.

SETKEY()

```
SETKEY(nInkeyCode, [bAction]) ⇒ bCurrentAction
```

nInkeyCode	is the INKEY() value of the key to be associated or queried.
bAction	specifies a code block that is automatically executed whenever the specified key is pressed during a wait state.

SETKEY() returns the action block currently associated with the specified key, or NIL if the specified key is not currently associated with a block.

SETMODE()

```
SETMODE(nRows, nCols) ⇒ lSuccess
```

nRows	is the number of rows in the desired display mode.
nCols	is the number of columns in the desired display mode.

SETMODE() is an environment function that attempts to change the mode of the display hardware to match the number of rows and columns specified. The change in screen size is reflected in the values returned by MAXROW() and MAXCOL().

SETPOS()

Set position

```
SETPOS(nRow, nCol) ⇒ NIL
```

nRow, ***nCol***	define the new screen position of the cursor. These values may range from 0, 0 to MAXROW(), MAXCOL().

SETPOS() is an environment function that moves the cursor to a new position on the screen. After the cursor is positioned, ROW() and COL() are updated accordingly.

SETPRC()

Set printer row column

```
SETPRC(nRow, nCol) ⇒ NIL
```

nRow	is the new PROW() value.
nCol	is the new PCOL() value.

SETPRC() is a printer function that sends control codes to the printer without changing the tracking of the printhead position.

SOUNDEX()

```
SOUNDEX(cString) ⇒ cSoundexString
```

cString	is the character string to convert.

SOUNDEX() returns a four-digit character string in the form A999.

SPACE()

```
SPACE(nCount) ⇒ cSpaces
```

nCount	is the number of spaces to return.

SPACE() returns a character string. If ***nCount*** is zero, SPACE() returns a null string ("").

SQRT()

```
SQRT(nNumber) ⇒ nRoot
```

nNumber	is a positive number to take the square root of.

SQRT() returns a numeric value calculated to double precision. The number of decimal places displayed is determined solely by SET DECIMALS regardless of SET FIXED. A negative ***nNumber*** returns zero.

STR()

String

```
STR(nNumber, [nLength], [nDecimals]) ⇒ cNumber
```

nNumber	is the numeric expression to convert to a character string.
nLength	is the length of the character string to return, including decimal digits, decimal point, and sign.
nDecimals	is the number of decimal places to return.

STR() returns ***nNumber*** formatted as a character string.

STRTRAN()

```
STRTRAN(cString, cSearch,
    [cReplace], [nStart], [nCount]) ⇒ cNewString
```

cString	is the character string or memo field to search.
cSearch	is the sequence of characters to locate.
cReplace	is the sequence of characters with which to replace ***cSearch***. If this argument is not specified, the specified instances of the search argument are replaced with a null string ("").
nStart	is the first occurrence that will be replaced. If this argument is omitted, the default is one.
nCount	is the number of occurrences to replace. If this argument is not specified, the default is all.

STRTRAN() returns a new character string with the specified instances of ***cSearch*** replaced with ***cReplace***.

STUFF()

```
STUFF(cString, nStart,
    nDelete, cInsert) ⇒ cNewString
```

cString	is the target character string into which characters are inserted and deleted.
nStart	is the starting position in the target string where the insertion/deletion occurs.
nDelete	is the number of characters to delete.
cInsert	is the string to insert.

STUFF() returns a copy of ***cString*** with the specified characters deleted and with ***cInsert*** inserted.

SUBSTR()

Sub string

```
SUBSTR(cString, nStart, [nCount]) ⇒ cSubstring
```

cString	is the character string from which to extract a substring.
nStart	is the starting position in ***cString***. If ***nStart*** is positive, it is relative to the leftmost character in ***cString***. If ***nStart*** is negative, it is relative to the rightmost character in the ***cString***.
nCount	is the number of characters to extract. If omitted, the substring begins at ***nStart*** and continues to the end of the string. If ***nCount*** is greater than the number of characters from ***nStart*** to the end of ***cString***, the extra is ignored.

SUBSTR() is a character function that extracts a substring from an-

other character string or memo field.

TIME()

```
TIME() ⇒ cTimeString
```

TIME() returns the system time as a character string in the form hh:mm:ss. hh is hours in 24-hour format, mm is minutes, and ss is seconds.

TIME() is a time function that displays the system time on the screen or prints it on a report.

TONE()

```
TONE(nFrequency, nDuration) ⇒ NIL
```

nFrequency	is a positive numeric value indicating the frequency of the tone to sound.
nDuration	is a positive numeric value indicating the duration of the tone measured in increments of 1/18 of a second. For example, an ***nDuration*** value of 18 represents one second.

For both arguments, noninteger values are truncated (not rounded) to their integer portion.

TRANSFORM()

```
TRANSFORM(exp, cSayPicture) ⇒ cFormatString
```

exp	is the value to format. This expression can be any valid data type except array, code block, and NIL.
cSayPicture	is a string of picture and template characters that describes the format of the returned haracter string.

TRANSFORM() converts ***exp*** to a formatted character string as defined by ***cSayPicture***.

TYPE()

```
TYPE(cExp) ⇒ cType
```

cExp	is a character expression whose type is to be determined. ***cExp*** can be a field, with or without the alias, a private or public variable, or an expression of any type.

TYPE() returns one of the following characters:

A	Array
B	Block
C	Character
D	Date
L	Logical
M	Memo
N	Numeric
O	Object
U	NIL, local, or static
UE	Error syntactical
UI	Error indeterminate

TYPE() is a system function that returns the type of the specified expression. TYPE() is like VALTYPE() but uses the macro operator (&) to determine the type of the argument. VALTYPE(), by contrast, evaluates an expression and determines the data type of the return value.

UPDATED()

```
UPDATED() ⇒ lChange
```

UPDATED() returns true (‘**.T.**’) if data in a GET is added or changed; otherwise, it returns false (‘**.F.**’).

UPPER()

```
UPPER(cString) ⇒ cUpperString
```

cString	is the character string to convert.

UPPER() returns a copy of ***cString*** with all alphabetical characters converted to uppercase. All other characters remain the same as in the original string.

USED()

```
USED() ⇒ lDbfOpen
```

USED() returns true (‘**.T.**’) if there is a database file in USE in the current work area; otherwise, it returns false (‘**.F.**’).

VAL()

Value

```
VAL(cNumber) ⇒ nNumber
```

cNumber	is the character expression to convert.

VAL() is a character conversion function that converts a character string containing numeric digits to a numeric value. When VAL() is executed, it evaluates ***cNumber*** until a second decimal point, the first non-numeric character, or the end of the expression is encountered.

VALTYPE()

Value type

```
VALTYPE(exp) ⇒ cType
```

exp	is an expression of any type.

VALTYPE() returns a single character representing the data type returned by ***exp***. VALTYPE() returns one of the following characters:

A	Array
B	Block
C	Character
D	Date
L	Logical
M	Memo
N	Numeric
O	Object
U	NIL

VALTYPE() is a system function that takes a single argument, evaluates it, and returns a one character string describing the data type of the return value.

YEAR()

```
YEAR(dDate) ⇒ nYear
```

dDate	is the date value to convert.

YEAR() returns the year of the specified date value including the century digits as a four-digit numeric value. The value returned is

not affected by the current DATE or CENTURY format. Specifying a null date (CTOD("")) returns zero.

nB functions

Some functions made into nB are available for macro use. Not all available functions are here documented.

ACCEPT()

```
ACCEPT( Field, [cMessage], [cHeader] ) ⇒ updatedField|NIL
```

It is a prompt function that shows ***cMessage*** asking to type something into ***Field***. It returns the updated data or NIL if [*Esc*] was pressed. The string ***cHeader*** is showed centered at the top window.

ACHOICE()

```
ACHOICE(nTop, nLeft, nBottom, nRight,
        acMenuItems,
        [alSelectableItems],
        [nInitialItem],
        [lButtons | aButtons]) ⇒ nPosition
```

nTop, ***nLeft***, ***nBottom***, ***nRight***	are the window coordinates.
acMenuItems	is an array of character strings to display as the menu items.
alSelectableItems	is a parallel array of logical values (one element for each item in ***acMenuItems***) that specify the selectable menu items. Elements can be logical values or character strings. If the element is a character string, it is evaluated as a macro expression which should evaluate to a logical data type. A value of false ('`.F.`') means that the corresponding menu item is not available, and a value of true ('`.T.`') means that it is available. By default, all menu items are available for selection.
nInitialItem	is the position in the ***acMenuItems*** array of the item that will be highlighted when the menu is initially displayed.
lButtons	if True means that default buttons will appear.
aButtons	is an array of buttons.
aButtons[***n***][1] == N	the nth button row position;
aButtons[***n***][2] == N	the nth button column position;
aButtons[***n***][3] == C	the nth button text;
aButtons[***n***][4] == B	the nth button code block.

ACHOICE() returns the numeric position in the ***acMenuItems*** array of the menu item selected. If no choice is made, ACHOICE() returns zero.

ACHOICEWINDOW()

```
ACHOICEWINDOW( acMenuItems, [cDescription],
        nTop, nLeft, nBottom, nRight,
        [alSelectableItems],
        [nInitialItem] ) ⇒ nPosition
```

acMenuItems	is an array of character strings to display as the menu items.
cDescription	is a header to be shown at the top of window.
nTop, ***nLeft***, ***nBottom***, ***nRight***	are the window coordinates.
alSelectableItems	is a parallel array of logical values (one element for each item in ***acMenuItems***) that specify the selectable menu items. Elements can be logical values or character strings. If the element is a character string, it is evaluated as a macro expression which should evaluate to a logical data type. A value of false ('`.F.`') means that the corresponding menu item is not available, and a value of true ('`.T.`') means that it is available. By default, all menu items are available for selection.
nInitialItem	is the position in the ***acMenuItems*** array of the item that will be highlighted when the menu is initially displayed.

ACHOICEWINDOW() calls ACHOICE() with a window border around the ACHOICE() screen area.

ALERTBOX()

```
ALERTBOX( cMessage, [aOptions] ) ⇒ nChoice
```

cMessage	is the message text displayed, centered, in the alert box. If the message contains one or more semicolons, the text after the semicolons is centered on succeeding lines in the dialog box.
aOptions	defines a list of up to 4 possible responses to the dialog box.

ALERTBOX() returns a numeric value indicating which option was chosen. If the [*Esc*] key is pressed, the value returned is zero. The ALERTBOX() function creates a simple modal dialog. The user can respond by moving a highlight bar and pressing the Return or SpaceBar keys, or by pressing the key corresponding to the first letter of the option. If ***aOptions*** is not supplied, a single "Ok" option is presented.

ALERTBOX() is similar to ALERT() but it accept mouse input.

ATB()

```
ATB( [nTop], [nLeft], [nBottom], [nRight],
     aArray, [nSubscript],
     [acColSayPic],
     [acColTopSep], [acColBodySep], [acColBotSep],
     [acColHead], [acColFoot],
     [abColValid],
     [abColMsg],
     [cColor], [abColColors],
     [lModify],
     [lButtons | aButtons] ) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom***, ***nRight***	defines the screen area where browse have to take place.
aArray	bidimensional array to be browsed.
nSubscript	starting array position.
acColSayPic	is the picture array.
acColTopSep	is the top separation array: default is chr(194)+chr(196).
acColBodySep	is the body separation array: default is chr(179).
acColBotSep	is the bottom separation array: default is chr(193)+chr(196).
acColHead	is the header array for every column.
acColFoot	is the footer array for every column.
abColValid	is the validation array that specify when a field is properly filled. The condition must be specified in code block format.

abColMsg	is the message array that permits to show information at the bottom of browse area. The array must be composed with code blocks which result with a character string.
cColor	is the color string: it may be longer than the usual 5 elements.
abColColors	is the color code block array. The code block receive as parameter the value contained inside the field and must return an array containing two numbers: they correspond to the two color couple from ***cColor***.
lModify	indicates whether the browse can modify data.
lButtons	if True, default buttons are displayed.
aButtons	array of buttons.
aButtons[***n***][1] N	the nth button row position;
aButtons[***n***][2] N	the nth button column position;
aButtons[***n***][3] C	the nth button text;
aButtons[***n***][4] B	the nth button code block.

This function starts the browse of a bidimensional array. Only arrays containing monodimensional array containing the same kind of editable data are allowed. The function can handle a maximum of 61 columns.

BCOMPILE()

«

```
BCOMPILE( cString ) ⇒ bBlock
```

Compiles the string ***cString*** and returns the code block ***bBlock***

BUTTON()

«

```
BUTTON( @aButtons,
      [nRow], [nCol], [cText], [cColor],
      [bAction] ) ⇒ NIL
```

aButtons	the array of buttons to be increased with a new button array.
nRow and ***nCol***	is the row and column starting position for the button string.
cText	is the text that make up the button.
cColor	is the color string.
bAction	is the code block associated to the button.

This function adds to ***aButtons*** a new button array. Please note that the button array added is compatible only with the READ() function and not the other function using array of buttons: the others do not have a color string.

COLORARRAY()

«

```
COLORARRAY( cColor ) ⇒ aColors
```

cColors	a color string to be translated into a color array.

This function transform a color string into a color array. The array has as many elements as the colors contained inside ***cColor*** string.

COORDINATE()

«

```
COORDINATE( [@nTop, @nLeft], @nBottom, @nRight,
      [cHorizontal], [cVertical] ) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom*** and ***nRight***	are the starting position of a window that is to be differently aligned.

cHorozontal	determinates the horizontal alignment: "L" all left; "l" middle left; "C" center; "c" center; "R" all right; "r" middle right.
cVertical	determinate the vertical alignment: "T" top; "t" up; "C" center; "c" center; "B" bottom; "b" down.

This function helps with the windows alignment recalculating and modifying ***nTop***, ***nLeft***, ***nBottom*** and ***nRight*** in the way to obtain the desired alignment.

COPYFILE()

«

```
COPYFILE( cSourceFile, cTargetFile|cDevice ) ⇒ NIL
```

cSourceFile	the source filename.
cTargetFile	the target filename.
cDevice	the target devicename.

This function copies the ***cSourceFile*** to ***cTargetFile*** or to ***cDevice***.

DBAPP()

«

```
DBAPP( cFileName, [acFields],
      [bForCondition], [bWhileCondition],
      [nNextRecords],
      [nRecord],
      [lRest],
      [cDriver] ) ⇒ NIL
```

cFileName	the filename containing data to append to the active alias.
acFields	array of fieldnames indicating the fields that should be updated on the active alias (default is all).
bForCondition	a code block containing the FOR condition to respect for the data append. Will be appended data that makes the evaluation of this code block True.
bWhileCondition	a code block containing the WHILE condition to respect for the data append. Will be appended data as long as the evaluation of this code block is True: the first time it becomes False, the data appending is terminated.
nNextRecord	if used, means that only the first ***nNextRecords*** will be appended.
nRecord	if used, means that that only the record ***nRecord*** will be appended.
lRest	this option is not available here also if the function saves a place for it.
cDriver	is the optional driver name to use to open the ***cFileName*** file.

This function is used to append data to the active alias using data from the ***cFileName*** file, that in this case is a '`.DBF`' file.

DBCLOSE()

«

```
DBCLOSE() ⇒ NIL
```

It is a substitution function of DBCLOSEALL() to use inside "compiled" macros, as a true DBCLOSEALL() will close the macro file too.

DBCONTINUE()

```
DBCONTINUE() ⇒ NIL
```

This function resumes a pending DBLOCATE().

DBCOPY()

```
DBCOPY( cFileName, [acFields],
        [bForCondition], [bWhileCondition],
        [nNextRecords],
        [nRecord],
        [lRest],
        [cDriver] ) ⇒ NIL
```

cFileName	the target filename for the data contained inside the active alias.
acFields	array of fieldnames indicating the fields that should be used from the active alias (default is all).
bForCondition	a code block containing the FOR condition to respect for the data copy. Will be copied the data that makes the evaluation of this code block True.
bWhileCondition	a code block containing the WHILE condition to respect for the data copy. Will be copied data as long as the evaluation of this code block is True: the first time it becomes False, the data copying is terminated.
nNextRecord	if used, means that only the first ***nNextRecords*** will be copied.
nRecord	if used, means that that only the record ***nRecord*** will be copied.
lRest	if used means that only the remaining records inside the active alias are copied.
cDriver	is the optional driver name to use to open the ***cFileName*** file.

This function is used to copy data to ***cFileName*** form the active alias.

DBCOPYSTRUCT()

```
DBCOPYSTRUCT( cDatabase, [acFields] ) ⇒ NIL
```

cDatabase	is a structure '.DBF' file that will be filled with structure information about the active alias.
acFields	is an array of fieldnames that should be taken into consideration.

This function creates a structure '.DBF' file copying the structure of the active alias.

DBCOPYXSTRUCT()

```
DBCOPYXSTRUCT( cExtendedDatabase ) ⇒ NIL
```

cExtendedDatabase	is a structure '.DBF' file that will be filled with structure information about the active alias, accepting extended structure informations.

This function creates a structure '.DBF' file copying the structure of the active alias. This function accept non-standard structure, that is, the extended structure available inside Clipper.

DBDELIM()

```
DBDELIM( lCopyTo, cFileName, [cDelimiter], [acFields],
         [bForCondition], [bWhileCondition],
         [nNextRecords], [nRecord], [lRest] ) ⇒ NIL
```

lCopyTo	if True the function work copying data to ***cFileName*** from the active alias, if False the function work appending data from ***cFileName*** to the active alias.
cFileName	the filename containing data to append to the active alias or to use as the target of the data copy from the active alias.
cDelimiter	the delimiter string (or character) used to separate fields inside ***cFileName***.
acFields	array of fieldnames indicating the fields of the active alias that should be taken into consideration (default is all).
bForCondition	a code block containing the FOR condition to respect. The operation will be made for all records that respect the condition.
bWhileCondition	a code block containing the WHILE condition to respect. The first time it becomes False, the operation is terminated.
nNextRecord	if used, means that only the first ***nNextRecords*** will be appended/copied.
nRecord	if used, means that that only the record ***nRecord*** will be appended/copied.
lRest	if used means that only the remaining records will be taken into consideration.

This function is used to append data to the active alias using data from the ***cFileName*** file or to copy data into ***cFileName*** using the active alias as the source. ***cFileName*** is a delimited ASCII file.

DBISTATUS()

```
DBISTATUS() ⇒ cDBInformations
```

This function returns the informations on the active alias in a text form.

DBISTRUCTURE()

```
DBISTRUCTURE() ⇒ cTextStructure | NIL
```

This function returns the structure information on the active alias in a text form.

DBJOIN()

```
DBJOIN( cAlias, cDatabase,
        [acFields], [bForCondition] ) ⇒ NIL
```

cAlias	the name of the alias to use to merge with records from the active alias.
cDatabase	the target '.DBF' filename.
acFields	the array of fieldnames which represent the projection of fields form both Aliases into the new '.DBF' file. If not specified, all fields from the primary work area are included in the target '.DBF' file.

This function creates a new database file by merging selected records and fields form two work areas (Aliases) based on a general condition. It works by making a complete pass through the secondary work area ***cAlias*** for each record in the primary work area (the active alias), evaluating the condition for each record in the secondary work area. When ***bForCondition*** is evaluated True, a new record is created in the target database file ***cDatabase*** using the fields specified from both work areas inside ***acFields***.

DBLABELFORM()

```
DBLABELFORM( cLabel, [lToPrinter], [cFile],
      [lNoConsole], [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest], [lSample] )
      ⇒ NIL
```

cLabel	is the name of the label file (.LBL) that contains the label format definition.
lToPrinter	if True, the output is copied to printer (‘LPT1:’).
cFile	if present, it is the name of a ASCII file where the output is copied.
lNoConsole	if True, the output is not sent to the console.
bForCondition	a code block containing the FOR condition to respect for label print. Only the records contained inside the active alias that respect the condition will be used for labels.
bWhileCondition	a code block containing the WHILE condition to respect for the label print. The first time that the condition is False, the label print terminates.
nNextRecord	if used, means that only the first ***nNextRecords*** will be used.
nRecord	if used, means that that only the record ***nRecord*** will be used.
lRest	if used means that only the remaining records inside the active alias will be used.
lSample	if True displays test labels as rows of asterisks.

This function prints labels to the console.

DBLIST()

```
DBLIST( [lToDisplay], abListColumns,
      [lAll],
      [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest],
      [lToPrinter], [cFileName] )
```

lToDisplay	if True the printout is sent to the console screen.
abListColumns	is an array of columns expressions to list.
lAll	if True prints all the records contained inside the active alias.
bForCondition	a code block containing the FOR condition to respect. Only the records contained inside the active alias that respect the condition will be used for list.
bWhileCondition	a code block containing the WHILE condition to respect. The first time that the condition is False, the list terminates.
nNextRecord	if used, means that only the first ***nNextRecords*** will be used.
nRecord	if used, means that that only the record ***nRecord*** will be used.
lRest	if used means that only the remaining records inside the active alias will be used.
lToPrinter	if True, the output is copied to printer (‘LPT1:’).
cFileName	if present, it is the name of a ASCII file where the output is copied.

This function prints a list of records to the console.

DBLOCATE()

```
DBLOCATE( [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest] ) ⇒ NIL
```

bForCondition	a code block containing the FOR condition to respect. Only the records contained inside the active alias that respect the condition will be taken into consideration.

bWhileCondition	a code block containing the WHILE condition to respect. The first time that the condition is False, the locate terminates.
nNextRecord	if used, means that only the first ***nNextRecords*** will be used.
nRecord	if used, means that that only the record ***nRecord*** will be used.
lRest	if used means that only the remaining records inside the active alias will be used.

This function searches sequentially for the first record matching the FOR and WHILE conditions. Once a DBLOCATE() has been issued you can resume the search from the current record pointer position with DBCONTINUE().

The WHILE condition and the scope (***nNextRecord***, ***nRecord*** and ***lRest***) apply only to the initial DBLOCATE() and are not operational for any subsequent DBCONTINUE() call.

DBOLDCREATE()

```
DBOLDCREATE( cDatabase, cExtendedDatabase,
      [cDriver], [lNew], [cAlias] ) ⇒ NIL
```

cDatabase	is the name of the new database file, with an optional drive and directory, specified as a character string. If specified without an extension (.dbf) is assumed.
cExtendedDatabase	is a ‘.DBF’ file containing the structure information of the file to create.
cDriver	specifies the replaceable database driver (RDD) to use to process the current work area. ***cDriver*** is the name of the RDD specified as a character expression.
lNew	if True the newly created ‘.DBF’ file is opened using the next available work area making it the current work area (the active alias).
cAlias	if ***lNew*** is set to True, this is the alias name to use to open the file.

This function is a old database function (superseded form DBCREATE()) that creates a database file from the structure information contained inside a structure file.

DBPACK()

```
DBPACK() ⇒ NIL
```

This function eliminates definitively the active alias records previously signed for deletion. It works only if the active alias is opened in exclusive mode.

DBSDF()

```
DBSDF( lCopyTo, cFileName, [acFields],
      [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest] ) ⇒ NIL
```

lCopyTo	if True the function works copying data to ***cFileName*** from the active alias, if False the function work appending data from ***cFileName*** to the active alias.
cFileName	the filename containing data to append to the active alias or to use as the target of the data copy from the active alias.
acFields	array of fieldnames indicating the fields of the active alias that should be taken into consideration (default is all).
bForCondition	a code block containing the FOR condition to respect. The operation will be made for all records that respect the condition.
bWhileCondition	a code block containing the WHILE condition to respect. The first time it becomes False, the operation is terminated.

nNextRecord	if used, means that only the first ***nNextRecords*** will be appended/copied.
nRercord	if used, means that that only the record ***nRecord*** will be appended/copied.
lReset	if used means that only the remaining records will be taken into consideration.

This function is used to append data to the active alias using data from the ***cFileName*** file or to copy data into ***cFileName*** using the active alias as the source. ***cFileName*** is a SDF ASCII file.

DBSORT()

«

```
DBSORT( cDatabase, [acFields],
      [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest] ) ⇒ NIL
```

cDatabase	the '.DBF' file to create.
acFields	the array of fields to be used to create the new sorted ***cDatabase*** file.
bForCondition	a code block containing the FOR condition to respect. Only the records contained inside the active alias that respect the condition will be taken into consideration.
bWhileCondition	a code block containing the WHILE condition to respect. The first time that the condition is False, the sort terminates.
nNextRecord	if used, means that only the first ***nNextRecords*** inside the active alias will be used.
nRecord	if used, means that that only the record ***nRecord*** will be used.
lRest	if used means that only the remaining records inside the active alias will be used.

Copy the active alias to a '.DBF' file in sorted order.

DBTOTAL()

«

```
DBTOTAL( cDatabase, bKey, [acFields],
      [bForCondition], [bWhileCondition],
      [nNextRecords], [nRecord], [lRest] ) ⇒ NIL
```

cDatabase	the '.DBF' file to create that will contain the copy of summarised records.
bKey	the code block key expression that should correspond to the key expression of the active index of the active alias.
acFields	the array of fields to be used to create the new ***cDatabase*** file.
bForCondition	a code block containing the FOR condition to respect. Only the records contained inside the active alias that respect the condition will be taken into consideration.
bWhileCondition	a code block containing the WHILE condition to respect. The first time that the condition is False, the sort terminates.
nNextRecords	if used, means that only the first ***nNextRecords*** inside the active alias will be used.
nRecord	if used, means that that only the record ***nRecord*** will be used.
lRest	if used means that only the remaining records inside the active alias will be used.

This function summarises records by key value to a '.DBF' file. It sequentially process the active alias scanning the specified scope of records. Records with the same key will be summarised inside the destination '.DBF' file. The value of numeric fields of records with the same key are added.

DBUPDATE()

«

```
DBUPDATE( cAlias, bKey, [lRandom], [bReplacement] )
```


cAlias	is the alias containing data to be used to update the active alias.
bKey	is a code block expression using information form the ***cAlias*** to obtain a key to refer to the active alias.
lRandom	if True, allows record in the ***cAlias*** to be in any order. In this case, the active alias must be indexed with the same key as ***bKey***.
bReplacement	is the code block that will be executed when records matches: it should contains the criteria for data update.

This function updates the active alias with data from another .DBF file.

Example:

```
dbUpdate( "INVOICE", {|| LAST}, .T.,;
    {|| FIELD->TOTAL1 := INVOICE->SUM1,;
    FIELD->TOTAL2 := INVOICE->SUM2 } )
```

DBZAP()

«

```
DBZAP() ⇒ NIL
```

This function erases immediately all the records contained inside the active alias.

DISPBOXCOLOR()

«

```
DISPBOXCOLOR( [nColorNumber], [cBaseColor] ) ⇒ cColor
```

nColorNumber	may be 1 or 2 and are the two color used to create shadowed borders. 1 is usually used for the left and top line; 2 is used for the right and bottom line.
cBaseColor	is the starting color string. The default is the actual color.

This function return a color string used for DISPBOXSHADOW() the function that create a shadowed border around a screen window.

DISPBOXSHADOW()

«

```
DISPBOXSHADOW( nTop, nLeft, nBottom, nRight,
      [cBoxString], [cColor1], [cColor2] ) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom*** and ***nRight***	are the screen coordinate where the box is to be displayed.
cBoxString	is the box string containing the character to use to build the box. Default is a single line box.
cColor1	is the color string to use for the left and top side of the box.
cColor2	is the color string to use for the right and bottom side of the box.

This function draws a screen box like DISPBOX() but allowing the variation of colors around the border to simulate a sort of shadow.

DIR()

«

```
DIR( [cFileSpec], [lDrives], [lDirs], [lFiles],
      [lNoDirReturn], [nSortColumn] ) ⇒ cPathname
```

cFileSpec	the filename or Pathname, also with wildcards, to be searched.
lDrives	true ('**.T.**') means: include drives letters.
lDirs	true ('**.T.**') means: include directory names.
lFiles	true ('**.T.**') means: include file names.
lNoRirReturn	true ('**.T.**') means: do not return the shown directory if [*Esc*] is used to exit.

nSortColumn	the column number to use to sort the list. The columns are: Name = 1, Size = 2, Date = 3, Time = 4, Attribute = 5. It is not possible to sort for extention.

It is a window function useful to search a file or a directory. The complete pathname of the selected file is returned.

DOC()

«

```
DOC( [cTextFileName] ) ⇒ NIL
```

cTextFileName	can contain the text file to open and edit; if empty, the editing of 'UNTITLED.TXT' will start.

It is the nB Text editor useful for small text files (less then 64K) and contains a complete menu that can be started with [F10].

Attention: doc() should not be used inside macros.

DOTLINE()

«

```
DOTLINE() ⇒ NIL
```

This function is a "dot" command line useful for calculations resolution. The dot-line content may be passed to the keyboard buffer.

DTEMONTH()

«

Date of month

```
DTEMONTH( nMonth, cLanguage ) ⇒ cMonth
```

nMonth	the month number.
cLanguage	the language name.

This function translates the ***nMonth*** number into the month name translated using the ***cLanguage*** language.

DTEWEEK()

«

Date of week

```
DTEWEEK( nWeek, cLanguage ) ⇒ cWeek
```

nWeek	is the week number (1 is Sunday, 7 is Saturday) to be translated into text.
cLanguage	is the language name into which the week must be expressed. At the moment it works only for Italian, so ***cLanguage*** can only contain "ITALIANO".

This function translates the week number into the week name translated using the ***cLanguage*** language.

EX()

«

Execute

```
EX( cFileMacro ) ⇒ nExitCode
```

Executes the macro file ***cFileName***. The extention must be specified.

cFileMacro may be the name of a "compiled" macro or a text macro file.

GET()

«

```
GET( @aGetList,
     [nTop], [nLeft],
     { |x| iif( pcount() > 0, Var := x, Var ) }
     [cGetPicture], [cColorString],
     [bPreExpression], [bValid] )
```

aGetList	is the get list array that will be increased with this get().
nTop and ***nLeft***	define the starting position of this get object on the screen.
Var	is the variable that is to be edited with this get. ***Var*** is in fact sent to the GET() function using a code block.
cGetPicture	is the get picture to use for ***Var***.
cColorString	is the color string to use for the get.
bPreExpression	is a code block that will be evaluated before the get object will became active. It must result True to obtain that the get object became active.
bValid	is a code block that will be evaluated after the get object is edited. It must result True to obtain that the get object may become inactive.

Create screen editing masks.

GVADD()

«

Get validation add

```
GVADD( @cField, cAdd ) ⇒ .T.
```

cField	the field to fill with more data.
cAdd	is the string to be added to the content of ***cField***.

This function is to be used inside GETs for pre/post validation, when a the content of a field should be added with more data.

cField is returned with the same length as before to avoid troubles with current and future GETs.

GVDEFAULT()

«

Get validation default

```
GVDEFAULT( @cField, cDefault ) ⇒ .T.
```

@cField	the field to check and if empty correct with ***cDefault***.
cDefault	is the default value to be used to replace ***cField***.

This function is to be used inside GETs for pre/post validation, when a field should have a default value.

cField is returned with the same length as before to avoid troubles with current and future GETs.

GVFILEDIR()

«

Get validation file directory

```
GVFILEDIR( @cWildName ) ⇒ .T.
```

cWildName	is the file name taken from the current get to be used for search with DIR().

This function is to be used inside GETs for pre validation: the ***cWildName*** is a file name with wild cards that can be searched with the DIR() function after that a specific key is pressed.

cWildName is returned with the same length as before to avoid troubles with current and future GETs.

GVFILEEXIST()

```
GVFILEEXIST( @cNameToTest, [cExtention] ) ⇒ lSuccess
```

@cNameToTest	is the file name taken from the current get to test for existence.
cExtention	is the normal extention of the file.

This function is to be used inside GETs for post validation: the file name have to exist.

cNameToTest is returned with the same length as before to avoid troubles with current and future GETs.

GVFILEEXTENTION()

```
GVFILEEXTENTION( @cName, cExt ) ⇒ .T.
```

@cName	the file name to be eventually corrected with file extention.
cExt	the file extention to use as default.

This function is to use inside GETs for pre/post validation, when the content of a field should contain a file name that should be corrected adding a default extention if not given from the user.

GVSUBST()

```
GVSUBST( @cField, cSubst ) ⇒ .T.
```

@cField	the field to be replaced with ***cSubst***.
cSubst	is the string to be used to replace the content of ***cField***.

This function is to use inside GETs for pre/post validation, when the content of a field should be replaced with other data.

cField is returned with the same length as before to avoid troubles with current and future GETs.

HTF()

```
HTF( [nInitialRecord] ) ⇒ NIL
```

nInitialRecord	is the record number where to start the Help Text File browse. Default is the actual record pointer.

This function browse a Help Text File that must be already opened and be the active alias.

ISFILE()

```
ISFILE( cName ) ⇒ lFileExists
```

cName	is the file name (with or without path) to be checked for existence.

This function returns true (‘**.T.**’) if the file ***cName*** exists. The difference between this function and the standard FILE() function is that ISFILE() checks for wildcards before. If ***cName*** contains wildcards, the result is false (‘**.F.**’).

ISWILD()

```
ISWILD( cName ) ⇒ lIsWild
```

cName	is the file name (with or without path) to be checked for wildcards presence.

This function returns true (‘**.T.**’) if ***cName*** contains wildcards.

ISMEMVAR()

```
ISMEMVAR( cName ) ⇒ lIsMemvar
```

cName	is the name of a possible memvar.

This function returns true (‘**.T.**’) if the ***cName*** is a declared Memvar.

ISCONSOLEON()

```
ISCONSOLEON() ⇒ lConsoleIsOn
```

This function returns true (‘**.T.**’) if the console will show the result of QOUT() and QQOUT().

ISPRINTERON()

```
ISPRINTERON() ⇒ lPrinterIsOn
```

This function returns true (‘**.T.**’) if the default printer will report the the result of QOUT() and QQOUT().

The default printer is ‘PRN:’ or ‘LPT1:’. If SET ALTERNATE TO is configured to send outputs to ‘LPT2:’ or another printer, the function will report false (‘**.F.**’).

KEYBOARD()

```
KEYBOARD( [cString] ) ⇒ NIL
```

This function stuff a string into the keyboard buffer.

LISTWINDOW()

```
LISTWINDOW( acMenuItem, [cDescription],
      [nTop], [nLeft], [nBottom], [nRight],
      [cColorTop], [cColorBody] ) ⇒ nPosition
```

acMenuItem	is the character array containing the list of choices.
cDescription	is the header to be shown at the top window.
nTop, ***nLeft***, ***nBottom***, ***nRight***	are the window coordinates.
cColorTop	is the color to use for window header and footer.
cColorBody	is the color to use for the window body that is the space where the text appears.

This function is an similar to achoice(), but it shows a header and footer, and it saves the screen, acting like a window.

MEMOWINDOW()

```
MEMOWINDOW( cVar, [cDescription], [nTop], [nLeft],
      [nBottom], [nRight], [cColorTop], [cColorBody],
      [lEditMode], [nLineLength], [nTabSize] ) ⇒ cVar
```

cVar	is the character field (variable) to be edited.
cDescription	is the header to be shown at the top window.
nTop, ***nLeft***, ***nBottom***, ***nRight***	are the window coordinates.
cColorTop	is the color to use for window header and footer.
cColorBody	is the color to use for the window body that is the space where the text appears.
lEditMode	is equivalent to memoedit().
nLineLength	is equivalent to memoedit().
nTabSize	is equivalent to memoedit().

This function lets you easily edit a long character field (memo) defining automatically a simple window and providing a simple help.

MEMPUBLIC()

```
MEMPUBLIC( cMemvarName | acMemvarNames ) ⇒ NIL
```

cMemvarName	is the name of the PUBLIC variable to create (max 10 characters).
acMemvarNames	is an array of PUBLIC variable names to create (max 10 characters).

Creates a PUBLIC variables or a group of variables.

MEMRELEASE()

```
MEMRELEASE( cMemvarName | acMemvarNames ) ⇒ NIL
```

cMemvarName	is the name of the PUBLIC variable to be released.
acMemvarNames	is an array of PUBLIC variable names to be released.

This function releases a previously created PUBLIC variables or a group of variables.

MEMRESTORE()

```
MEMRESTORE( cMemFileName, [lAdditive] ) ⇒ NIL
```

cMemFileName	the memory file (.MEM) to load from disk.
lAdditive	if True causes memory variables loaded from the memory file to be added to the existing pool of memory variables. If False, the existing memory variables are automatically released.

Retrieve memory variables form a memory file (.MEM).

MEMSAVE()

```
MEMSAVE( cMemFileName, [cSkeleton], [lLike] ) ⇒ NIL
```

cMemFileName	the memory file (.MEM) where public variables should be saved.
cSkeleton	the skeleton mask for defining a group of variables. Wildcard characters may be used: _*_ and _?_.
lLike	if True, the variables grouped with ***cSkeleton*** are saved, else only the other variables are saved.

Saves memory variables to a memory file (.MEM).

MENUPROMPT()

```
MENUPROMPT( @aoGet,
      [nRow], [nCol],
      [cPrompt], [bBlock] ) ⇒ NIL
```

aoGet	is an array of get objects where a new get is added by MENUPROMPT(). These gets are read only.
nRow and ***nCol***	are the screen coordinates where the menu prompt will appear.
cPrompt	is the menu prompt string.
bBlock	is the code block to execute when the cursor is on the current menu prompt. It is usually a code block that shows a message somewhere on the screen.

This function should substitute the @...PROMPT command and handle the mouse.

MENUTO()

```
MENUTO( aoGet, nPos ) ⇒ nChoice
```

aoGet	array of get objects.
nPos	starting position to be edited.

Like MENU TO. It returns the selected menu item created with MENUPROMPT(). It supports the mouse.

MESSAGELINE()

```
MESSAGELINE( [cMessage], [cColor], [nPosTop], [nPosLeft] )
      ⇒ NIL
```

aMessage	the message to be displayed.
cColor	the color string.
nPosTop and ***nPosLeft***	the starting position where the string message would appear on the screen. Default values are respectively ROW() and COL().

MESSAGELINE() is a function that display a message on the screen on the selected position. If ***cMessage*** is NIL, the message is eliminated from screen restoring the previous screen content.

MOUSESCRSAVE()

```
MOUSESCRSAVE( [nTop], [nLeft], [nBottom], [nRight] )
      ⇒ cSavedScreen
```

nTop, ***nLeft***, ***nBottom*** and ***nRight***	are the screen coordinates that will be to save the screen.

This function works line SAVESCREEN() but it hide the mouse cursor before a screen save is made.

MOUSESCRRESTORE()

```
MOUSESCRRESTORE( [nTop], [nLeft], [nBottom], [nRight],
      [cScreen] ) ⇒ cSavedScreen
```

nTop, ***nLeft***, ***nBottom*** and ***nRight***	are the screen coordinates where the saved screen will be restored.
cScreen	is the previously saved screen to restore.

This function works line RESTSCREEN() but it hide the mouse cursor before a screen restore is made.

PICCHRMAX()

```
PICCHRMAX( [nCol], [nMaxCol] ) ⇒ cPictureString
```

nCol	is the starting position on the screen for the get field.
nMaxCol	is the end position on the screen of the get field.

This function is useful when a character field is to be used on a get object. The generated picture will be the of the maximum possible extention, eventually with scroll.

QUIT()

```
QUIT() ⇒ NIL
```

Terminates program execution.

READ()

```
READ( aoGet, [nPos], [aButtons], [lReadOnly] )
      ⇒ lUpdated
```

aoGet	is the array of get objects.
nPos	is the starting position.
aButtons	is the array of buttons.
lReadOnly	if True, get fields cannot be modified; the default value is False.

This function is made to substitute the READMODAL() allowing the use of the mouse. The array ***aButtons*** is made with the help of the function BUTTON().

RF()

```
RF( cFRMName,
    [bForCondition], [bWhileCondition],
    [nNext], [nRecord], [lRest], [lPlain],
    [cbHeading], [lBeforeEject], [lSummary],
    [lDate], [acExtra] ) ⇒ NIL
```

cFRMName	the form (.FRM) file to use to print the active alias.
bForCondition	code block for the FOR condition.
bWhileCondition	code block for the WHILE condition.
nNext	see REPORT FORM.
nRecord	see REPORT FORM
lRest	see REPORT FORM
lPlain	if true ('`.T.`'), force the print in a simple way.
cbHeading	additional header in character or code block form. If a code block is sent, the final result must be a character string.
lBeforeEject	if true ('`.T.`'), force a form feed before the print.
lSummary	if true ('`.T.`'), force a summary print only.
lDate	if false ('`.F.`'), force the print without date at the top of page.
acExtra	a character array that may be used for translating standard printed report form words and to add vertical and horizontal separations. The default value of acExtra is: ***acExtra***[1] "Page No." ***acExtra***[2] "** Subtotal **" ***acExtra***[3] "* Subsubtotal *" ***acExtra***[4] "*** Total ***" ***acExtra***[5] " " vertical column separation ***axExtra***[6] "" horizontal separation: no separation.

This function does the same work of REPORT FORM or __ReportForm or dbReportForm, but it prints where qout() and qqout() print.

RPT()

```
RPT( cText ) ⇒ NIL
```

This function prints the text contained into ***cText*** using print commands. This function accepts other parameters here not described, as they are not to be used for macro purpose. The printing is made using QOUT() and QQOUT(), this way it is sensible to the "alternate" file definition.

RPTMANY()

```
RPTMANY( cText, [bWhileCondition], [bForCondition] )
      ⇒ NIL
```


cText	is the text to be printed.
bWhileCondition	is a code block for a WHILE condition to respect for the records to print.
bForCondition	is a code block for a FOR condition to respect for the records to print.

This function prints the text contained into ***cText*** many times: one for every record contained into the active alias.

RPTTRANSLATE()

```
RPTTRANSLATE( cText ) ⇒ cTranslatedText
```

This function translates once ***cText*** replacing variables with memvars or Fields.

RUN()

```
RUN( cCommand ) ⇒ NIL
```

This function start execution of ***cCommand*** in a DOS session. It works only if there is enough available memory.

SAY()

```
SAY( nTop, nLeft, Expr,
     [cSayPicture], [cColorString] ) ⇒ NIL
```

nTop and ***nLeft***	define the starting position on the screen where the ***Expr*** should be displayed.
nLeft	is an expression that will be solved and displayed.
cSayPicture	is the picture to use to display ***Expr***.
cColorString	is the color string to use.

This function displays the result of ***Expr*** on the screen on the desired position.

SETCOLORSTANDARD()

```
SETCOLORSTANDARD( [nColor], [cColor|acColor] )
      ⇒ cPreviousColor|acPreviousColor
```

nColor	is the color number to take into consideration: 0 All colors 1 Base 2 Menu 3 Head 4 Body (Say - Get) 5 Button (Mouse buttons) 6 Message 7 Alert
cColor	the color string to be associated with ***nColor***.
acColor	it the color array

This function is a way to handle colors inside the application. The functions that display something use a default color depending on what they does. These colors may be changed with SETCOLORSTANDARD(), all together or only one.

SETFUNCTION()

```
SETFUNCTION( nFunctionKey, cString ) ⇒ NIL
```

nFunctionKey	the number of the function key (1=F1, 12=F12) to be assigned.
cString	the character string.

This function assigns a character string to a function key (obsolete).

SETMOUSE()

```
SETMOUSE( [lShow] ) ⇒ lPrevious
```

lShow	True shows the mouse cursor, False hide the mouse cursor, NIL reports only the status.

This function is made to show, hide or report only the mouse cursor status.

SETOUTPUT()

```
SETOUTPUT( [cPeriperal | aPeripheral] )
      ⇒ aPrevious_Output_Peripherals
```

cPeripheral	is the new output peripheral for qout() and qqout() functions.
aPeripheral	are the new output peripherals configurations for qout() and qqout() functions.

nB is organised in the way to have only one output peripheral at the time. This function help to make order inside SET CONSOLE, SET PRINTER and SET ALTERNATE.

If ***cPeripheral*** contains:

"CON"
SET CONSOLE is set to ON,
SET PRINTER is set to OFF,
SET ALTERNATE is set to OFF;

"PRN"
SET CONSOLE is set to OFF,
SET PRINTER is set to ON,
SET ALTERNATE is set to OFF;

"LPT1"
same as "PRN";

otherwise
SET CONSOLE is set to OFF,
SET PRINTER is set to OFF,
SET ALTERNATE is set to ON,
SET ALTERNATE TO is set to ***cPeripheral***.

aPeripheral is organised this way:

aPeripheral[1] = _SET_CONSOLE

aPeripheral[2] = _SET_PRINTER

aPeripheral[3] = _SET_ALTERNATE

aPeripheral[4] = _SET_ALTFILE

aPeripheral[5] = _SET_EXTRA

aPeripheral[6] = _SET_EXTRAFILE

This function is necessary because SET ALTERNATE alone is not enough to print on the screen when the peripheral name is "CON" or to print on the printer when the peripheral name is "PRN" or "LPT1". In fact, in the first case, ROW() and COL() will not be updated, in the second case, PROW() and PCOL() will not be updated.

This function returns an array organised in the same way as ***aPeripheral*** is, that shows the active output configuration.

SETRPTEJECT()

```
SETRPTEJECT( [lbEject] ) ⇒ lPreviousEjectMode
```

This function is used to set the eject mode after every page print for RPT(). If single sheet paper is used, then SETRPTEJECT(.T.) must be set; for continuous paper, SETRPTEJECT(.F.) is correct. The default value is .F..

lbEject	logical or code block, is the eject mode to set. Default is no change, the starting value is '**.F.**'

SETRPTLINES()

```
SETRPTLINES() ⇒ nRemainingLines
```

This function is used to report the number of lines available before the completion of the page print for RPT().

SETVERB()

Set verbose

```
SETVERB( cSpecifier, [xNewSetting], [lOpenMode] )
      ⇒ xPreviousValueSet
```

cSpecifier	a word that defines the kind of set is going to be considered.
xNewSetting	is the new value to set up.
lOpenMode	used only for some kind of set.

This function is analogue to SET() but it uses a character string (with ***cSpecifier***) and not a number to select the set. This is made to make easier the work with macros.

cSpecifier may contain:

"EXACT"
"FIXED"
"DECIMALS"
"DATEFORMAT"
"EPOCH"
"PATH"
"DEFAULT"
"EXCLUSIVE"
"SOFTSEEK"
"UNIQUE"
"DELETED"
"CANCEL"
"TYPEAHEAD"
"COLOR"
"CURSOR"
"CONSOLE"
"ALTERNATE"
"ALTFILE"
"DEVICE"
"EXTRA"
"EXTRAFILE"
"PRINTER"
"PRINTFILE"
"MARGIN"
"BELL"
"CONFIRM"
"ESCAPE"
"INSERT"
"EXIT"
"INTENSITY"
"SCOREBOARD"
"DELIMITERS"
"DELIMCHARS"
"WRAP"
"MESSAGE"
"MCENTER"

SETVERB("EXACT") (obsolete)

```
SETVERB( "EXACT", [lExact] ) ⇒ lPrevious
```

If ***lExact*** is True, it forces exact comparison of character strings, including length. If it is False, character strings are compared until the left string length is exhausted; that is that "" (the null string) is equal to any other string.

Please note that the == operator is a comparison operator for exact match and using it, SETVERB("EXACT", '`.F.`') will not work.

The starting value is True; the recommended value is True.

SETVERB("FIXED")

«

```
SETVERB( "FIXED", [lFixed] ) ⇒ lPrevious
```

If ***lFixed*** contains True, numeric values are displayed ever with a fixed number of decimal digits, depending on the value set by SETVERB("DECIMALS").

The starting value is False.

The recommended value is False: if you have to display a fixed number of decimal digits it is better to define a good display picture.

SETVERB("DECIMALS")

«

```
SETVERB( "DECIMALS", [nDecimals] ) ⇒ nPrevious
```

nDecimals is the number of digits to display after the decimal position. This set is enabled of disabled with SETVERB("FIXED").

The starting value is 8.

SETVERB("DATEFORMAT")

«

```
SETVERB( "DATEFORMAT", [cDateFormat] ) ⇒ cPrevious
```

cDateFormat is a character expression that specifies the date format.

The starting value is "dd/mm/yyyy".

Some date format examples:

AMERICAN	"mm/dd/yyyy"
ANSI	"yyyy.mm.dd"
BRITISH	"dd/mm/yyyy"
FRENCH	"dd/mm/yyyy"
GERMAN	"dd.mm.yyyy"
ITALIAN	"dd-mm-yyyy"
JAPAN	"yyyy/mm/dd"
USA	"mm-dd-yyyy"

SETVERB("EPOCH")

«

```
SETVERB( "EPOCH", [nYear] ) ⇒ nPrevious
```

nYear specifies the base year of 100-year period in which all dates containing only two year digits are assumed to fall.

The starting value is 1900.

SETVERB("PATH")

«

```
SETVERB( "PATH", [cPath] ) ⇒ cPrevious
```

cPath identifies the paths that nB uses when searching for a file not found in the current directory. The list of paths can be separated by commas or semicolons.

The starting value is "".

SETVERB("DEFAULT")

«

```
SETVERB( "DEFAULT", [cPath] ) ⇒ cPrevious
```

cPath identifies the default disk drive and directory.

The starting value is "".

SETVERB("EXCLUSIVE")

«

```
SETVERB( "EXCLUSIVE", [lExclusive] ) ⇒ lPrevious
```

If ***lPath*** is True, the default database (.DBF) file open is made in exclusive mode; in the other case, in shared mode.

The starting value is True.

SETVERB("SOFTSEEK")

«

```
SETVERB( "SOFTSEEK", [lSoftSeek] ) ⇒ lPrevious
```

If ***lSoftSeek*** is True, if a DBSEEK() index search fails, the record pointer is moved to the next record with a higher key. If it is False, in case of a DBSEEK() index search failure, the record pointer is moved at EOF().

The starting value is False.

SETVERB("UNIQUE") (obsolete)

«

```
SETVERB( "UNIQUE", [lUnique] ) ⇒ lPrevious
```

If ***lUnique*** is True, during creation or update of '`.DBF`' indexes, if two or more records are found with the same key, only the first record will be included inside the index.

If ***lUnique*** is False, duplicated record keys are allowed.

The starting value is False.

SETVERB("DELETED")

«

```
SETVERB( "DELETED", [lDeleted] ) ⇒ lPrevious
```

If ***lDeleted*** is True, record signed for deletion are not filtered, that is, these are still normally visible as they were not deleted. In the other case, they hare (in most cases) hidden to the user.

The starting value is False.

SETVERB("CANCEL")

«

```
SETVERB( "CANCEL", [lCancel] ) ⇒ lPrevious
```

If ***lCancel*** is True, enables [*Alt c*] and [*Ctrl Break*] as termination keys. In the other case, not.

The starting value is True.

SETVERB("TYPEAHEAD")

«

```
SETVERB( "TYPEAHEAD", [nTypeAhead] ) ⇒ nPrevious
```

nTypeAhead is the number of keystrokes the keyboard buffer can hold from a minimum of zero to a maximum of 4096.

The starting value is 15.

SETVERB("COLOR")

«

```
SETVERB( "COLOR", [cColorString] ) ⇒ cPrevious
```

nColorString defines the normal screen colors. There are five couple of colors, but only three are really operative:

standard	This is the standard color used for screen output.
enhanced	This is the color used for highlighted screen output.
border	Normally unused.
background	Normally unused.
unselected	This is the color used for GET fields without focus.

The default color string is "BG+/B,N/W,N/N,N/N,W/N" that is:

standard	bright Cyan on Blue
enhanced	Black on White
border	Black on Black
background	Black on Black
unselected	White on Black

The following table explains the use of letters inside the color string. Note that the plus sign (+) means high intensity, the star (*) means blink and that + and * can be allowed only to the first letter inside a couple.

Color	Letter	Monochrome
Black	N, Space	Black
Blue	B	Underline
Green	G	White
Cyan	BG	White
Red	R	White
Magenta	RB	White
Brown	GR	White
White	W	White
Gray	N+	Black
Bright Blue	B+	Bright Underline
Bright Green	G+	Bright White
Bright Cyan	BG+	Bright White
Bright Red	R+	Bright White
Bright Magenta	RB+	Bright White
Bright Brown	GR+	Bright White
Bright White	W+	Bright White
Black	U	Underline
Inverse Video	I	Inverse Video
Blank	X	Blank

SETVERB("CURSOR")

«

```
SETVERB( "CURSOR", [lCursor] ) ⇒ lPrevious
```

If ***lCursor*** is True, the cursor is showed, else it is hidden.

The starting value is True.

SETVERB("CONSOLE")

«

```
SETVERB( "CONSOLE", [lConsole] ) ⇒ lPrevious
```

If ***lConsole*** is True, the output of console commands is displayed on the screen, else it is not.

The starting value is True.

SETVERB("ALTERNATE")

«

```
SETVERB( "ALTERNATE", [lAlternate] ) ⇒ lPrevious
```

If ***lAlternate*** is True, the output of console commands is send also to a standard ASCII text file.

The starting value is False.

SETVERB("ALTFILE")

«

```
SETVERB( "ALTFILE", [cAlternateFilename], [lAdditive] )
      ⇒ cPrevious
```

If SETVERB("ALTERNATE") is True, the output of the console is send also to ***cAlternateFilename***, a standard ASCII file.

If ***lAdditive*** is True, the output is appended to the ASCII file if it already exists, else it is erased first.

SETVERB("DEVICE")

«

```
SETVERB( "DEVICE", [cDevice] ) ⇒ cPrevious
```

cDevice is the name of the device where SAY() will display its output.

The starting value is "SCREEN", the alternative is "PRINTER".

The recommended value is "SCREEN".

SETVERB("EXTRA")

«

```
SETVERB( "EXTRA", [lExtra] ) ⇒ lPrevious
```

If ***lExtra*** is True, the output of console commands is send also to a standard ASCII text file.

The starting value is False.

SETVERB("EXTRAFILE")

«

```
SETVERB( "EXTRAFILE", [cExtraFilename], [lAdditive] )
      ⇒ cPrevious
```

If SETVERB("EXTRA") is True, the output of the console is send also to ***cExtraFilename***, a standard ASCII file.

If ***lAdditive*** is True, the output is appended to the ASCII file if it already exists, else it is erased first.

SETVERB("PRINTER")

«

```
SETVERB( "PRINTER", [lPrinter] ) ⇒ lPrevious
```

If ***lPrinter*** is True, the output of console commands is also printed, else it is not.

The starting value is False.

SETVERB("PRINTFILE")

«

```
SETVERB( "PRINTFILE", [cPrintFileName] ) ⇒ cPrevious
```

cPrintFileName is the name of the printer peripheral name.

The starting value is "" (null string).

SETVERB("MARGIN")

«

```
SETVERB( "MARGIN", [nPageOffset] ) ⇒ nPrevious
```

nPageOffset is the positive number of column to be used as a left margin for all printer output.

The starting value is 0.

SETVERB("BELL")

«

```
SETVERB( "BELL", [lBell] ) ⇒ lPrevious
```

If ***lBell*** is True, the sound of the bell is used to get the attention of the user when some wrong actions are made.

The starting value is False.

SETVERB("CONFIRM")

```
SETVERB( "CONFIRM", [lConfirm] ) ⇒ lPrevious
```

If ***lConfirm*** is False, the GET is simply terminated typing over the end of the get field; in the other case (True), the GET is terminated only pressing an "exit key". The starting value is True.

SETVERB("ESCAPE")

```
SETVERB( "ESCAPE", [lEscape] ) ⇒ lPrevious
```

If ***lEscape*** is True, the [*Esc*] key is enabled to be a READ exit key, in the other case not.

The starting value is True.

The recommended value is True.

SETVERB("INSERT")

```
SETVERB( "INSERT", [lInsert] ) ⇒ lPrevious
```

If ***lInsert*** is True, the data editing is in INSERT mode, in the other case, it is in OVERWRITE mode.

The starting value is True.

SETVERB("EXIT")

```
SETVERB( "EXIT", [lExit] ) ⇒ lPrevious
```

If ***lExit*** is True, [Up] and [Down] key may be used as exit key when the cursor is (respectively) on the first or on the last GET field. In the other case not.

The starting value is False.

The recommended value is False.

SETVERB("INTENSITY")

```
SETVERB( "INTENSITY", [lIntensity] ) ⇒ lPrevious
```

If ***lIntensitiy*** is True, the display of standard and enhanced display colors are enabled. In the other case, only standard colors are enabled.

The starting value is True.

The recommended value is True.

SETVERB("SCOREBOARD")

```
SETVERB( "SCOREBOARD", [lScoreboard] ) ⇒ lPrevious
```

If ***lScoreboard*** is True, the display of messages from READ() and MEMOREAD() is allowed; in the order case not.

The starting value is False.

The recommended value is False: nB do not support scoreboard.

SETVERB("DELIMITERS")

```
SETVERB( "DELIMITERS", [lDelimiters] ) ⇒ lPrevious
```

If ***lDelimiters*** is True, GET variables appear on the screen delimited with the delimiter symbols. In the other case, GET variables are not delimited this way, but only with the use of different colors.

The starting value is False.

The recommended value is False: the use of delimiters creates one more trouble when designing a screen mask.

SETVERB("DELIMCHARS")

```
SETVERB( "DELIMCHARS", [cDelimterCharacters] ) ⇒ cPrevious
```

cDelimterCharacters are the delimiter characters used to delimit a GET field when SETVERB("DELIMITERS") is True.

The starting value is "::".

SETVERB("WRAP")

```
SETVERB( "WRAP", [lWrap] ) ⇒ lPrevious
```

If ***lWrap*** is True, the wrapping of the highlight in MENUs should be active, but this option is actually not active and all works as it is False.

The starting value is False.

SETVERB("MESSAGE")

```
SETVERB( "MESSAGE", [nMessageRow] ) ⇒ nPrevious
```

nMessageRow is the row number where the @..PROMPT message line should appear on the screen. This option is not supported.

The starting value is 0.

SETVERB("MCENTER")

```
SETVERB( "MCENTER", [lMessageCenter] ) ⇒ lPrevious
```

If ***lMessageCenter*** is True, the @..PROMPT message line should appear centered on the screen. This option is not supported.

The starting value is False.

STRADDEXTENTION()

```
STRADDEXTENTION( cName, cExt ) ⇒ cCompleteName
```

cName	the file name (with or without path) that is probably without extention.
cExt	the extention that must be added to ***cName*** if it has not one.

This function check ***cName*** for the presence of an extention. It it has not one, ***cExt*** will be added.

STRCUTEXTENTION()

```
STRCUTEXTENTION( cName ) ⇒ cName
```

cName	the file name (with or without path) that is probably with extention.

This function check ***cName*** for the presence of an extention. It it has one, the extention is removed.

STRDRIVE()

```
STRDRIVE( cName ) ⇒ cDrive
```

cName	the file name (with or without path) that contains the drive letter.

This function tries to extract the drive letter information from ***cName***.

STREXTENTION()

«

```
STREXTENTION( cName ) ⇒ cExtention
```

cName	the file name (with or without path) that contains an extention.

This function tries to extract the extention information from ***cName***.

STRFILE()

«

```
STRFILE( cName ) ⇒ cFileName
```

cName	the file name with or without path.

This function tries to extract the file name without path from ***cName***.

STRFILEFIND()

«

```
STRFILEFIND( cName, cPath ) ⇒ cFileName
```

cName	the file name or pathname containing the file name to search inside the ***cPath*** list.
cPath	a list of paths separated with semicolon (just like Dos does), where ***cFile*** should be searched.

If your file is to be found on different possible positions, this function search the first place where the file is found and returns a valid pathname to that file.

STRGETLEN()

«

```
STRGETLEN( xExpr, cPicture ) ⇒ nFieldLength
```

xExpr	a generic expression.
cPicture	the picture string.

This function returns the length of field when using ***xExpr*** with ***cPicture***.

STRLISTASARRAY()

«

```
STRLISTASARRAY( cList, [cDelimiter] ) ⇒ aList
```

cList	a character string containing a list separated with ***cDelimiter***.
cDelimiter	the delimiter used to separate the elements contained inside the list.

This function transform a character string list into an array.

STROCCURS()

«

```
STROCCURS( cSearch, cTarget ) ⇒ nOccurrence
```

cSearch	the search string to find inside ***cTarget***.
cTarget	the string to be searched for the presence of ***cSearch***.

This function returns the number of occurrence that ***cSearch*** is contained inside ***cTarget***.

STRPARENT()

«

```
STRPARENT( cName ) ⇒ cParentPath
```

cName	the pathname.

This function tries to return a parent path from ***cName***.

STRPATH()

«

```
STRPATH( cName ) ⇒ cPath
```

cName	the pathname.

This function tries to extract the path from ***cName***.

STRTEMPPATH()

«

```
STRTEMPPATH() ⇒ cTempPath
```

This function returns a temporary path searching for possible definitions inside the environmental variables.

STRXTOSTRING()

«

```
STRXTOSTRING( xVar, [cType] ) ⇒ cTrasformed_to_string
```

xVar	is the data of any type to be converted into string.
cType	is the type of the data contained inside ***xVar***.

This function returns ***xVar*** transformed into a character string.

TB()

«

```
TB( [nTop], [nLeft], [nBottom], [nRight],
    [acCol], [acColSayPic],
    [acColTopSep], [acColBodySep], [acColBotSep],
    [acColHead], [acColFoot],
    [alColCalc],
    [abColValid],
    [abColMsg],
    [cColor], [abColColors],
    [nFreeze],
    [lModify],
    [lAppend],
    [lDelete],
    [lButtons | aButtons] ) ⇒ NIL
```

nTop, ***nLeft***, ***nBottom***, ***nRight*** defines the screen area where browse have to take place.

acCol	is the columns array to be included into the browse.
acColSayPic	is the picture array.
acColTopSep	is the top separation array: default is chr(194)+chr(196).
acColBodySep	is the body separation array: default is chr(179).
acColBotSep	is the bottom separation array: default is chr(193)+chr(196).
acColHead	is the header array for every column.
acColFoot	is the footer array for every column.
alColCalc	is the array that identify the calculated column (not editable). True (‘**.T.**’) means calculated.
abColValid	is the validation array that specify when a field is properly filled. The condition must be specified in code block format.
abColMsg	is the message array that permits to show information at the bottom of browse area. The array must be composed with code blocks which result with a character string.
cColor	is the color string: it may be longer than the usual 5 elements.
abColColors	is the color code block array. The code block receive as parameter the value contained inside the field and must return an array containing two numbers: they correspond to the two color couple from ***cColor***.

nFreeze	indicates the number of columns to be left frozen on the left side.
lModify	indicates whether the browse can modify data.
lDelete	indicates whether the browse can delete and recall records.
lButtons	if True, default buttons are displayed.
aButtons	array of buttons.
aButtons[***n***][1] N	the ***n***th button row position;
aButtons[***n***][2] N	the ***n***th button column position;
aButtons[***n***][3] C	the ***n***th button text;
aButtons[***n***][4] B	the ***n***th button code block.

This function, called without parameters, starts the browse of the active alias, and if relations are established, the browse includes also related data.

Please note that due to an unresolved problem, the field names contained inside ***acCol*** should better contain also the alias (ALIAS->FIELD_NAME). See also the examples.

TEXT()

```
TEXT( cText ) ⇒ NIL
```

Shows the text contained into ***cText***.

TGLINSERT()

```
TGLINSERT() ⇒ NIL
```

Toggle the global insert mode and the cursor shape.

TIMEX2N()

```
TIMEX2N( [nHH], [nMM], [nSS] ) ⇒ nTime
```

nHH	is the number of hours.
nMM	is the number of minutes.
nSS	is the number of seconds.

This function calculate the "time number" that is a number representing days and/or portion of a day: 1 is 1 day or 24 hours, 0.5 is 12 hours, and so on.

TIMEN2H()

```
TIMEN2H( nTime ) ⇒ nHours
```

nTime	is the "time number" that is a number representing days and/or portion of a day: 1 is 1 day or 24 hours, 0.5 is 12 hours, and so on.

This function returns the integer number of hours contained inside ***nTime***.

TIMEN2M()

```
TIMEN2M( nTime ) ⇒ nMinutes
```

nTime	is the "time number" that is a number representing days and/or portion of a day: 1 is 1 day or 24 hours, 0.5 is 12 hours, and so on.

This function returns the integer number of minutes contained inside ***nTime*** after subtracting the hours.

TIMEN2S()

```
TIMEN2S( nTime ) ⇒ nSeconds
```

nTime	is the "time number" that is a number representing days and/or portion of a day: 1 is 1 day or 24 hours, 0.5 is 12 hours, and so on.

This function returns the number of seconds (with eventual decimals) contained inside ***nTime*** after subtracting the hours and the minutes.

TRUESETKEY()

```
TRUESETLEY( nInkeyCode, bAction ) ⇒ .T.
```

This function is equivalent to SETKEY() but it returns always '**.T.**'

WAITFILEEVAL()

```
WAITFILEEVAL( lClose ) ⇒ .T.
```

Shows a wait bar calling WAITPROGRESS() for operation on records of a database.

If there is no index active, it is equivalent to WAITPROGRES(RECNO()/LASTREC()).

if an index is active, this cannot work, so an increment for each call is made: WAITPROGRES((nIncrement++)/LASTREC()).

This function must be closed calling it with the ***lClose*** parameter to true ('**.T.**'). This way, internal counters are closed and WAITPROGRESS() is closed too.

WAITFOR()

```
WAITFOR( [cMessage] ) ⇒ NIL
```

Shows ***cMessage*** until it is called again. The wait window is closed when called without parameter or with NIL.

WAITPROGRESS()

```
WAITPROGRESS( [nPercent] ) ⇒ .T.
```

Shows a wait bar on the screen top depending on the value contained into ***nPercent***. ***nPercent*** starts form 0 and ends to 1 (100%). If a value of one or more, or NIL is passed, the wait window is closed.

Normal command substitution

Clipper works only with functions and commands that are converted into function using the 'STD.CH'. Here are described some command replacement that can be used also with nB macros.

?

```
? [exp_list]
```

```
qout([exp_list])
```

```
?? [exp_list]
```

```
qqout([exp_list])
```

@BOX

```
@ nTop, nLeft, nBottom, nRight BOX cnBoxString [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, [cnBoxString], [cColorString])
```

@TO

```
@ nTop, nLeft TO nBottom, nRight DOUBLE [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, 2 [,cColorString])
```

```
@ nTop, nLeft TO nBottom, nRight [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, 1 [,cColorString])
```

```
@ nTop, nLeft CLEAR [TO nBottom, nRight]
```

```
scroll([nTop], [nLeft], [nBottom, nRight])
```

```
setpos(nRow, nCol)
```

@GET

```
@ nTop, nLeft GET Var [PICTURE cGetPicture] [COLOR cColorString] [WHEN lPreExpression] ↩
↪[VALID lPostExpression]
```

```
setpos(nTop, nLeft)
```

```
aadd( GetList, _GET_( Var, "Var", cGetPicture, [{|| lPostExpression}], ↩
↪[{|| lPreExpression}] ):display() ) atail(GetList):colorDisp(cColor
```

@SAY

```
@ nTop, nLeft SAY exp [COLOR cColorString]
```

```
devpos(nTop, nLeft)
```

```
devout(exp [, cColorString])
```

```
@ nTop, nLeft SAY exp PICTURE cSayPicture [COLOR cColorString]
```

```
devpos(nTop, nLeft)
```

```
devoutpic(exp, cSayPicture, [cColorString])
```

APPEND

```
APPEND BLANK
```

```
dbappend()
```

CLEAR

```
CLEAR
```

```
Scroll()
```



```
SetPos(0,0)
```

```
ReadKill(.T.)
```

```
GetList := {}
```

```
CLEAR GETS
```

```
ReadKill(.T.)
```

```
GetList := {}
```

```
CLEAR SCREEN | CLS
```

```
Scroll()
```

```
SetPos(0,0)
```

CLOSE

```
CLOSE
```

```
dbCloseArea()
```

```
CLOSE idAlias
```

```
idAlias->( dbCloseArea() )
```

```
CLOSE ALTERNATE
```

```
Set(19, "")
```

```
CLOSE DATABASES
```

```
dbCloseAll()
```

```
CLOSE INDEXES
```

```
dbClearIndex()
```

COMMIT

```
COMMIT
```

```
dbCommitAll()
```

COUNT

```
COUNT TO idVar [FOR lForCondition] [WHILE lWhileCondition] [NEXT nNextRecords] ↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||idVar:=idVar+1}, {||lForCondition}, {||lWhileCondition}, ↩
↪nNextRecords, nRecord, lRest )
```

DEFAULT

```
DEFAULT xVar TO xDefaultValue
```

```
DEFAULT( @xVar, xDefaultValue ) ⇒ xVar
```

DELETE

```
DELETE
```

```
dbDelete()
```

```
DELETE [FOR lForCondition] [WHILE lWhileCondition] [
NEXT nNextRecords]↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||dbDelete()}, {||lForCondition}, {||lWhileCondition},↩
↪nNextRecords, nRecord, lRest )
```

```
DELETE FILE xcFile
```

```
ferase( cFile )
```

EJECT

```
EJECT
```

```
qqout( chr(13) )
```

ERASE

```
ERASE xcFile
```

```
ferase( cFile )
```

FIND

```
FIND xcSearchString
```

```
dbSeek( cSearchString )
```

GO

```
GO[TO] nRecord
```

```
dbgoto(nRecord)
```

```
GO[TO] BOTTOM
```

```
dbGoBottom()
```

```
GO[TO] TOP
```

```
dbgotop()
```

INDEX ON

```
INDEX ON expKey TO xcIndexName [UNIQUE] [FOR lForCondition]↩
↪[WHILE lWhileCondition] [[EVAL lEvalCondition] [EVERY nRecords
]] [ASCENDING|DESCENDING]
```



```
ordCondSet( [cForCondition], [bForCondition], , [bWhileCondition
],↩
↪[bEvalCondition], [nRecords], RECNO(), , , , lDescending )
```

```
ordCreate( cIndexName, , cExpKey, bExpKey, lUnique )
```

READ

```
READ
```

```
ReadModal(GetList)
```

```
GetList := {}
```

```
READ SAVE
```

```
ReadModal(GetList)
```

RECALL

```
RECALL
```

```
dbRecall()
```

```
RECALL [FOR lForCondition] [WHILE lWhileCondition] [
NEXT nNextRecords]↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||dbRecall()}, {||lForCondition}, {||lWhileCondition},↩
↪nNextRecords, nRecord, lRest )
```

REINDEX

```
REINDEX [EVAL lEvalCondition] [EVERY nRecords]
```

```
ordCondSet(, , , , [bEvalCondition], [nRecords]
, , , , , , , , )
```

```
ordListRebuild()
```

RENAME

```
RENAME xcOldFile TO xcNewFile
```

```
frename( cOldFile, cNewFile )
```

REPLACE

```
REPLACE idField1 WITH exp1 [, idField2 WITH exp2…]↩
↪[FOR lForCondition] [WHILE lWhileCondition] [NEXT nNextRecords]
↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {|| idField1 := exp1 [, idField2 := exp2…]},↩
↪{||lForCondition}, {||lWhileCondition}, nNextRecords,↩
↪nRecord, lRest )
```

```
REPLACE idField1 WITH exp1
```

```
idField1 := exp1
```

RESTORE

```
RESTORE SCREEN FROM cScreen
```

```
restscreen( 0, 0, Maxrow(), Maxcol(), cScreen )
```

SAVE

```
SAVE SCREEN TO cScreen
```

```
cScreen := savescreen( 0, 0, maxrow(), maxcol() )
```

SEEK

```
SEEK expSearch [SOFTSEEK]
```

```
dbSeek( expSearch [, lSoftSeek] )
```

SELECT

```
SELECT xnWorkArea | idAlias
```

```
dbSelectArea( nWorkArea | cIdAlias )
```

SET

```
SET ALTERNATE TO xcFile [ADDITIVE]
```

```
Set( 19, cFile, lAdditive )
```

```
SET ALTERNATE ON | OFF | xlToggle
```

```
Set( 18, "ON" | "OFF" | lToggle )
```

```
SET BELL ON | OFF | xlToggle
```

```
Set( 26, "ON" | "OFF" | lToggle )
```

```
SET COLOR | COLOUR TO (cColorString)
```

```
SetColor( cColorString )
```

```
SET CONFIRM ON | OFF | xlToggle
```

```
Set( 27, "ON" | "OFF" | lToggle )
```

```
SET CONSOLE ON | OFF | xlToggle
```

```
Set( 17, "ON" | "OFF" | lToggle )
```

```
SET CURSOR ON | OFF | xlToggle
```

```
SetCursor( 1 | 0 | iif( lToggle, 1, 0 ) )
```

```
SET DATE FORMAT [TO] cDateFormat
```

```
Set( 4, cDateFormat )
```



```
SET DECIMALS TO
```

```
Set( 3, 0 )
```

```
SET DECIMALS TO nDecimals
```

```
Set( 3, nDecimals )
```

```
SET DEFAULT TO
```

```
Set( 7, "" )
```

```
SET DEFAULT TO xcPathspec
```

```
Set( 7, cPathspec )
```

```
SET DELETED ON | OFF | xlToggle
```

```
Set( 11, "ON" | "OFF" | lToggle )
```

```
SET DELIMITERS ON | OFF | xlToggle
```

```
Set( 33, "ON" | "OFF" | lToggle )
```

```
SET DELIMITERS TO [DEFAULT]
```

```
Set( 34, "::" )
```

```
SET DELIMITERS TO cDelimiters
```

```
Set( 34, cDelimiters )
```

```
SET DEVICE TO SCREEN | PRINTER
```

```
Set( 20, "SCREEN" | "PRINTER" )
```

```
SET EPOCH TO nYear
```

```
Set( 5, nYear )
```

```
SET ESCAPE ON | OFF | xlToggle
```

```
Set( 28, "ON" | "OFF" | lToggle )
```

```
SET EXACT ON | OFF | xlToggle
```

```
Set( 1, "ON" | "OFF" | lToggle )
```

```
SET EXCLUSIVE ON | OFF | xlToggle
```

```
Set( 8, "ON" | "OFF" | lToggle )
```

```
SET FILTER TO
```

```
dbclearfilter()
```

```
SET FILTER TO lCondition
```

```
dbsetfilter( bCondition, cCondition )
```

```
SET FIXED ON | OFF | xlToggle
```

```
Set( 2, "ON" | "OFF" | lToggle )
```

```
SET INDEX TO [xcIndex [, xcIndex1... ] ]
```

```
ordListClear()
ordListAdd( cIndex )
ordListAdd( cIndex1 )
…
```

```
SET INTENSITY ON | OFF | xlToggle
```

```
Set( 31, "ON" | "OFF" | lToggle )
```

```
SET KEY nInkeyCode [TO]
```

```
SetKey( nInkeyCode, NIL )
```

```
SET KEY nInkeyCode TO [idProcedure]
```

```
SetKey( nInkeyCode, { |p, l, v| idProcedure(p, l, v)} )
```

```
SET MARGIN TO
```

```
Set( 25, 0 )
```

```
SET MARGIN TO [nPageOffset]
```

```
Set( 25, nPageOffset )
```

```
SET MESSAGE TO
```

```
Set( 36, 0 )
```

```
Set( 37, .F. )
```

```
SET MESSAGE TO [nRow [CENTER | CENTRE]]
```

```
Set( 36, nRow )
```



```
Set( 37, lCenter )
```

```
SET ORDER TO [nIndex]
```

```
ordSetFocus( nIndex )
```

```
SET PATH TO
```

```
Set( 6, "" )
```

```
SET PATH TO [xcPathspec [, cPathspec1... ] ]
```

```
Set( 6, cPathspec [, cPathspec1... ] )
```

```
SET PRINTER ON | OFF | xlToggle
```

```
Set( 23, "ON" | "OFF" | lToggle )
```

```
SET PRINTER TO
```

```
Set( 24, "" )
```

```
SET PRINTER TO [xcDevice|xcFile [ADDITIVE]]
```

```
Set( 24, cDevice|cFile, lAdditive )
```

```
SET RELATION TO
```

```
dbclearrelation()
```

```
SET RELATION TO [expKey1 INTO xcAlias1]
     [, [TO] expKey2 INTO xcAlias2...]
     [ADDITIVE]
```

```
if !lAdditive
     dbClearRel()
end
dbSetRelation( cAlias1, {|| expKey1}, ["expKey1"] )
dbSetRelation( cAlias2, {|| expKey2}, ["expKey1"] )
```

```
SET SCOREBOARD ON | OFF | xlToggle
```

```
Set( 32, "ON" | "OFF" | lToggle )
```

```
SET SOFTSEEK ON | OFF | xlToggle
```

```
Set( 9, "ON" | "OFF" | lToggle )
```

```
SET TYPEAHEAD TO nKeyboardSise
```

```
Set( 14, nKeyboardSise )
```

```
SET UNIQUE ON | OFF | xlToggle
```

```
Set( 10, "ON" | "OFF" | lToggle )
```

```
SET WRAP ON | OFF | xlToggle
```

```
Set( 35, "ON" | "OFF" | lToggle )
```

SKIP

```
SKIP [nRecords] [ALIAS idAlias|nWorkArea]
```

```
[idAlias|nWorkArea -> ]( dbSkip([nRecords]) )
```

STORE

```
STORE value TO variable
```

```
variable := value
```

SUM

```
SUM nExp1 [, nExp2…] TO idVar1 [, idVar2…] [FOR lForCondition]
↩
↪[WHILE lWhileCondition] [NEXT nNextRecords] [RECORD nRecord]
[REST] [ALL]
```

```
dbeval( {||idVar1:=idVar1+nExp1 [, idVar2:=idVar2+nExp2…] },↩
↪{||lForCondition}, {||lWhileCondition}, nNextRecords, nRecord, lRest )
```

UNLOCK

```
UNLOCK
```

```
dbUnlock()
```

```
UNLOCK ALL
```

```
dbUnlockAll()
```

USE

```
USE
```

```
dbclosearea()
```

```
USE [xcDatabase]↩
↪[INDEX xcIndex1 [, xcIndex2…] [ALIAS xcAlias] [EXCLUSIVE|
SHARED] ↩
↪[NEW] [READONLY] [VIA cDriver]]
```

```
dbUseArea( [lNewArea], [cDriver], cDatabase, [cAlias], [lShared
], [lReadOnly] )
[dbSetIndex( cIndex1 )]
[dbSetIndex( cIndex2 )]
…
```


nB command substitution functions

Inside nB there are many functions made only in substitution to other Clipper commands.

GET

```
@ nTop, nLeft GET Var
     [PICTURE cGetPicture]
     [COLOR cColorString]
     [WHEN lPreExpression]
     [VALID lPostExpression]
```

```
Get( @aGetList,
     [nTop], [nLeft],
     { |x| iif( pcount() > 0, Var := x, Var ) }
     [cGetPicture], [cColorString],
     [bPreExpression], [bValid] )
```

aGetList	is the get list array that will be increased with this get().

SAY

```
@ nTop, nLeft SAY exp
     PICTURE cSayPicture
     [COLOR cColorString]
```

```
Say( nTop, nLeft, cVar, [cSayPicture], [cColorString] )
```

APPEND FROM

```
APPEND FROM xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     [VIA xcDriver]
```

```
dbApp( cFileName, [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords],
     [nRecord],
     [lRest],
     [cDriver] )
```

```
APPEND FROM xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     DELIMITED xcDelimiter
```

```
dbDelim( .f., cFileName, [cDelimiter], [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```

```
APPEND FROM xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     SDF
```

```
dbSDF( .f., cFileName, [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```

CONTINUE

```
CONTINUE
```

```
dbContinue()
```

COPY

```
COPY FILE xcSourceFile TO xcTargetFile | xcDevice
```

```
CopyFile( cSourceFile, cTargetFile | cDevice )
```

```
COPY STRUCTURE [FIELDS idField_list]
     TO xcDatabase
```

```
dbCopyStruct( cDatabase, [acFields] )
```

```
COPY STRUCTURE EXTENDED
     TO xcExtendedDatabase
```

```
dbCopyXStruct( cExtendedDatabase )
```

```
COPY TO xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     [VIA xcDriver]
```

```
dbCopy( cFileName, [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords],
     [nRecord],
     [lRest],
     [cDriver] )
```

```
COPY TO xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     DELIMITED xcDelimiter
```

```
dbDelim( .t., cFileName, [cDelimiter], [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```

```
COPY TO xcFile
     [FIELDS idField_list]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     SDF
```

```
dbSDF( .t., cFileName, [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```


CREATE

```
CREATE xcDatabase
     FROM xcExtendedDatabase
     [NEW]
     [ALIAS cAlias]
     [VIA cDriver]
```

```
dbOldCreate( cDatabase, cExtendedDatabase,
     [cDriver], [lNew], [cAlias] )
```

JOIN

```
JOIN WITH xcAlias TO xcDatabase
     [FOR lCondition] [FIELDS idField_list]
```

```
dbJoin( cAlias, cDatabase,
     [acFields], [bForCondition] )
```

KEYBOARD

```
KEYBOARD cString
```

```
Keyboard( [cString] ) ⇒ NIL
```

LABEL FORM

```
LABEL FORM xcLabel
     [TO PRINTER]
     [TO FILE xcFile]
     [NOCONSOLE]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     [SAMPLE]
```

```
dbLabelForm( cLabel, [lToPrinter], [cFile],
     [lNoConsole], [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest], [lSample] )
```

LIST

```
LIST exp_list
     [TO PRINTER]
     [TO FILE xcFile]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     [OFF]
```

```
dbList( [lToDisplay], abListColumns,
     [lAll],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest],
     [lToPrinter], [cFileName] )
```

LOCATE

```
LOCATE [scope] FOR lCondition
     [WHILE lCondition]
```

```
dbLocate( [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```

PACK

```
PACK
```

```
dbPack()
```

PUBLIC

```
PUBLIC idMemvar
```

```
MemPublic( cMemvarName|acMemvarNames )
```

QUIT

```
QUIT
```

```
Quit()
```

RELEASE

```
RELEASE idMemvar
```

```
MemRelease( cMemvarName|acMemvarNames )
```

REPORT FORM

```
REPORT FORM xcReport
     [TO PRINTER]
     [TO FILE xcFile]
     [NOCONSOLE]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
     [PLAIN | HEADING cHeading]
     [NOEJECT] [SUMMARY]
```

```
RF( cForm,
     [bForCondition], [bWhileCondition],
     [nNext], [nRecord], [lRest], [lPlain],
     [cbHeading], [lBeforeEject], [lSummary],
     [lDate], [acExtra] ) ⇒ NIL
```

RESTORE FROM

```
RESTORE FROM xcMemFile [ADDITIVE]
```

```
MemRestore( cMemFileName, [lAdditive] )
```

RUN

```
RUN xcCommandLine
```

```
Run( cCommand )
```

SAVE TO

```
SAVE TO xcMemFile
     [ALL [LIKE|EXCEPT skeleton]]
```

```
MemSave( cMemFileName, [cSkeleton], [lLike] )
```

SET FUNCTION

```
SET FUNCTION nFunctionKey TO cString
```

```
SetFunction( nFunctionKey, cString )
```


SORT

```
SORT TO xcDatabase
     ON idField1 [/[A|D][C]]
     [, idField2 [/[A|D][C]] …]
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
```

```
dbSort( cDatabase, [acFields],
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord], [lRest] )
```

TOTAL

```
TOTAL ON expKey
     [FIELDS idField_list] TO xcDatabase
     [scope]
     [WHILE lCondition]
     [FOR lCondition]
```

```
dbTotal( cDatabase, bKey, [acFields,
     [bForCondition], [bWhileCondition],
     [nNextRecords], [nRecord]], [lRest] )
```

UPDATE

```
UPDATE FROM xcAlias
     ON expKey [RANDOM]
     REPLACE idField1 WITH exp
     [, idField2 WITH exp …]
```

```
dbUpdate( cAlias, bKey, [lRandom], [bReplacement] )
```

Example:

```
dbUpdate( "INVOICE", {|| LAST}, .T.,;
     {|| FIELD->TOTAL1 := INVOICE->SUM1,;
     FIELD->TOTAL2 := INVOICE->SUM2 } )
```

ZAP

```
ZAP
```

```
dbZap()
```

RPT: the nB print function

The function RPT() helps to print ASCII file containing Memvars, Fields and print commands. RPT() is accessible from the DOC() menu.

Memvars and fields

As usual with standard word processors, variables are written delimited with "<" (Alt+174) and ">" (Alt+175).

Inside these delimiters can find place character Memvars, character Fields and functions giving a character result.

The RPT() function generates a public variable n_Lines that contains the available lines inside the actual sheet. Every time a line is written, this value is reduced, until a new page is reached and then it will start again from the maximum value. It is useful to read this variable to determinate if there is enough space or it is better to change page.

Commands

The function RPT() recognise some print commands. These commands starts with the asterisk (*) symbol. This means that "*" is a print command prefix.

It follows the command syntax.

***COMMAND**

```
*COMMAND
      cStatement
      cStatement
      …
*END
```

The lines contained inside *COMMAND - *END are executed with the nB macro interpreter.

***DBSKIP**

```
*DBSKIP [nSkip]
```

It Executes a dbskip() on the active alias.

***FOOT**

```
*FOOT
      cFooter
      cFooter
      …
*END
```

The lines contained inside *FOOT - *END are printed each time at the bottom of pages.

***HEAD**

```
*HEAD
      cHeader
      cHeader
      …
*END
```

The lines contained inside *HEAD - *END are printed each time at the top of pages.

***IF**

```
*IF lCondition
    …
    …
*END
```

If the condition ***lCondition*** is true, the lines contained inside *IF - *END are printed.

***INSERT**

```
*INSERT cFileName
```

Includes the text contained into the file ***cFileName***.

***LEFT**

```
*LEFT nLeftBorder
```

The ***nLeftBorder*** is the number of column to be left blank as a left border.

***LPP**

```
*LPP nLinesPerPage
```

It determinates the page length expressed in lines. After printing the ***nLinesPerPage***th line, a form feed is sent.

***NEED**

```
*NEED nLinesNeeded
```

If the available lines are less then ***nLinesNeeded***, the follwing text will be printed on the next page.

***PA**

```
*PA
```

Jumps to a new page.

***REM**

```
*REM | *COMMENT [comment_line]
```

It adds a comment that will not be printed.

***WHILE**

```
*WHILE lCondition
    …
    …
*END
```

The lines contained inside *WHILE - *END are printed as long as ***lCondition*** is true.

Examples

It follows some example of text to be printed with the RPT() function. Example's lines are numbered. Line numbers must not be part of a real RPT text files.

PAGE DEFINITION

Margins are defined with *HEAD, *FOOT and *LEFT commands. In the following example is defined:

```
Top           2 lines;

Bottom        2 lines;

Left          10 characters.
```

The right margin is not defined as it depends on the lines length that will be printed.

The only considered page dimension is the height, *LPP (lines per page):

```
Page height   66 lines.
```

Here starts the example:

```
001  *lpp 66
002  *head
003
004
005  *end
006  *foot
007
008
009  *end
010  *left 10
011  ... text text text
012  ... test text text
...
```

At line 001 is defined the page height in lines. At line 002 is defined the header; it contains two empty lines (003 and 004) which will be printed at the top of every page. At line 006 starts the footer definition that contains two empty lines (007 and 008) that will be printed at the end of every page. At line 010 is defined the space on the left that will be added to every line printed. From line 011 starts the normal text.

HEADER AND FOOTER

The commands *HEAD and *FOOT are used to define the top and bottom border if they contains empty lines, it these lines are not empty, they became real head and foot.

The dimensions are as it follows:

```
Top           6 lines (should be one inch);

Bottom        6 lines;

Left          10 characters (should be an inch).

Page height   66 lines (should be 11 inch).
```

At position 0.5 in (after 3 lines) a one line header appears.

```
001  *lpp 66
002  *head
003
004
005
006  ------------------- MYFILE.TXT ------------------
007
008
009  *end
010  *foot
011
012
013
014
015
016
017  *end
018  *left 10
019  ... text text text
020  ... test text text
 ...
```

At line 006 (the fourth header line) a text appears. It will be printed on every page at the absolute fourth page line.

CODE INSERTION

Pieces of code can be inserted inside *COMMAND - *END. It can be useful to make complicated reports.

The following example declares a public variable used to number pages.

```
001  *command
002  mempublic("PageNo")
003  PageNo := 0
004  *end
005  *lpp 66
006  *head
007  *command
008  PageNo := PageNo +1
009  *end
010
011
012  *end
013  *foot
014
015                                Page <PageNo>
016
017  *end
018  *left 10
019  ... text text text
020  ... test text text
 ...
```

At line 001 starts a *COMMAND definition: lines 002 and 003 will be interpreted from the function EX(), the nB interpreter. These lines define a public variable and initialize it at 0. This variable will be use to count pages.

At line 007, inside the header (nested), start another *COMMAND definition that contains an increment for the "PageNo" variable. As the header is read and "executed" for every new page, and that before the footer, the variable "PageNo" will contain the right page number.

At line 015, inside the footer, a reference to "PageNo" appears. Here will be printed the page number.

A more complicated example can be found in ‘`ADDRESS.TXT`’ the RPT text file used for the ADDRESS.& macro examples.

How can I...

nB is a little bit complicated as it may do many things. Here are some examples.

Create a UDF function

UDF means User Defined Function. Inside nB there isn’t the possibility to create functions, but there is an alternative: code blocks.

Create a big code block

A code block cannot be longer than 254 characters, as any other instruction inside nB.

So, there is no way to make a bigger code block, but a code block can call another code block, and so on. For example:

```
mempublic( { "first", "second", "third" } )
first := {|| eval( second, "hello" ) }
second := {|x| eval( third, x ) }
third := {|x| alertbox( x ) }
eval( first )
```


This stupid example simply will show the alert box containing the word "hello".

The source files

The nB source is composed of four files:

‘`NB.PRG`’	The main source file containing essentially the nB menu.
‘`REQUEST.PRG`’	Contains a link to all Clipper standard functions.
‘`STANDARD.PRG`’	Contains the most important standard functions.
‘`EXTRA.PRG`’	Contains some extra function not absolutely necessary during macro execution.

The file ‘`REQUEST.PRG`’ source file generates some warnings because not all functions listed there are directly called form nB. Don’t worry about that warning message.

Different ‘`.RMK`’ (rmake) files are included to compile nB differently, including/excluding some program parts, for example to obtain a runtime executor.

[1] This is the original documentation of nanoBase 1997, with minor modifications, that appeared originally at ‘`http://www.geocities.com/SiliconValley/7737/nb.htm`’.

Step 1: try to compile with the /P parameter 1259
Step 2: understand well the use of code blocks 1259
Step 3: understand the object programming 1260
Classes and methods 1260
Class definition 1260
Object creation 1260
Instantiating an object 1260
The ‘‘send’’ symbol 1261
More about objects 1261
Step 4: understand the get object 1261
Step 5: trying to stop using commands 1262
?/?? 1263
@...BOX 1263
@...GET 1263
@...SAY 1263
@...TO 1263
APPEND 1264
APPEND FROM 1264
CLEAR 1264
CLOSE 1265
COMMIT 1265
CONTINUE 1265
COPY 1265
COUNT 1266
CREATE 1266
DEFAULT 1266
DELETE 1267
EJECT 1267
ERASE 1267
FIND 1267
GO 1267
INDEX ON 1267
JOIN 1268
KEYBOARD 1268
LABEL FORM 1268
LIST 1268
LOCATE 1268
PACK 1268
QUIT 1268
READ 1269
RECALL 1269
REINDEX 1269
RELEASE 1269
RENAME 1270
REPLACE 1270
REPORT FORM 1270
RESTORE 1270
RESTORE FROM 1270
RUN 1270
SAVE SCREEN TO 1270
SAVE TO 1270
SEEK 1271
SELECT 1271
SET 1271
SKIP 1276

SORT 1276
STORE 1276
SUM 1276
TOTAL ON 1276
UNLOCK 1276
UPDATE FROM 1276
USE 1277
ZAP 1277
Step 6: free yourself from STD.CH - /U 1277
Step 7: take control over all include files 1277

A different way to program using Clipper 5.2 without commands, that is, without the file 'STD.CH'.[1]

Step 1: try to compile with the /P parameter 1259
Step 2: understand well the use of code blocks 1259
Step 3: understand the object programming 1260
Classes and methods 1260
Class definition 1260
Object creation 1260
Instantiating an object 1260
The ''send'' symbol 1261
More about objects 1261
Step 4: understand the get object 1261
Step 5: trying to stop using commands 1262
?/?? 1263
@...BOX 1263
@...GET 1263
@...SAY 1263
@...TO 1263
APPEND 1264
APPEND FROM 1264
CLEAR 1264
CLOSE 1265
COMMIT 1265
CONTINUE 1265
COPY 1265
COUNT 1266
CREATE 1266
DEFAULT 1266
DELETE 1267
EJECT 1267
ERASE 1267
FIND 1267
GO 1267
INDEX ON 1267
JOIN 1268
KEYBOARD 1268
LABEL FORM 1268
LIST 1268
LOCATE 1268
PACK 1268
QUIT 1268
READ 1269
RECALL 1269
REINDEX 1269
RELEASE 1269

RENAME 1270
REPLACE 1270
REPORT FORM 1270
RESTORE 1270
RESTORE FROM 1270
RUN 1270
SAVE SCREEN TO 1270
SAVE TO 1270
SEEK 1271
SELECT 1271
SET 1271
SKIP 1276
SORT 1276
STORE 1276
SUM 1276
TOTAL ON 1276
UNLOCK 1276
UPDATE FROM 1276
USE 1277
ZAP 1277
Step 6: free yourself from STD.CH - /U 1277
Step 7: take control over all include files 1277

Clipper 5.2,[2] as the xBase tradition imposes, is not an ordered, clear, simple programming language. The question is: which is the right way to write a Clipper program? If the intention is not to make a xBase program, but a Clipper program, maybe it can be decided that it is better to use Clipper without commands.

Step 1: try to compile with the /P parameter

Supposing to compile the file 'TEST.PRG' this way:

```
C:\>CLIPPER TEST.PRG /P [Enter]
```

It generates a preprocessed output file ('test.PPO'), that is a source file without comments, where commands are translated into real Clipper instructions. That is, all the '**#COMMAND**' substitution are executed and the translation is sent to the '.PPO' file. It may be difficult to read this file the first time.

Step 2: understand well the use of code blocks

The code block is a small piece of executable program code that can be stored inside a variable, or can be used as a literal constant. The good of it, is that pieces of code may be sent to functions.

A code block is something like a little user defined function where only a sequence of expressions (functions and/or assignments) may appear: no loops, no conditional structures.

A code block may receive arguments and return a value after execution, just like a function. The syntax is the following, where curly brackets are part of the code block:

```
{ | [argument_list] | exp_list }
```

That is: the ***argument_list*** is optional; the ***exp_list*** may contain one or more expressions separated with a comma.

For example, calling the following code block will give the string ''hello world'' as result.

```
{ || "hello world" }
```

The following code block requires a numeric argument and returns the number passed as argument incremented:

```
{ | n | n+1 }
```

The following code block requires two numeric arguments and returns the sum of the two square radix:

```
{ | nFirst, nSecond | SQRT(nFirst) + SQRT(nSecond) }
```

But code blocks may contain more expressions and the result of the execution of the code block is the result of the last expression. The following code block executes in sequence some functions and gives ‘‘hello world’’ as a result.

```
{ | a, b | functionOne(a), functionTwo(b), "hello world" }
```

To start the execution of a code block a function is used: ‘**EVAL()**’. For example, a code block is assigned to a variable and then executed.

```
B := { || "hello world" }
EVAL( B ) == "hello world"
```

Another example with one parameter.

```
B := { | n | n+1 }
EVAL( B, 1 ) == 2
```

Another example with two parameters.

```
B := { | nFirst, nSecond | SQRT(nFirst) + SQRT(nSecond) }
EVAL( B, 2, 4 ) == 20
```

And so on.

Step 3: understand the object programming

Clipper 5.2 do not permit to create objects, but it gives some good objects to use: ‘**GET**’ and ‘**TBROWSE**’. Before starting to clean programming from commands, it is necessary to understand how to use well the Clipper objects.

Classes and methods

A class defines the structure of a ‘‘black box’’, that is a data container; a method is an action to make on a piece of data contained inside the black box. There is no way to reach the data contained inside the black box without a method.

The black box can be called object.

The methods may be seen as they where special functions which interact with a specific piece of data contained inside the object.

Class definition

Supposing that Clipper permits to define classes (unluckily only predefined classes can be used), the hypothetical syntax could be:

```
CLASS ClassName [FROM ParentClass]
    VAR Var1 [, Var2[, ...VarN]]
    METHOD {method_definition_1} [, ...{method_definition_n} ]
ENDCLASS
```

This way, the class defines a group of variables and a group of method to use with these variables.

Object creation

The presence of classes permits to create objects: the black boxes.

```
Variable_name := ClassName
```

This way, a variable contains (is) an object. Please note that inside Clipper, an object may be generated also from a function, that is, a function can return an object. This way the example can be:

```
Variable_name := classfunction( … )
```

The next problem is to handle this object.

Instantiating an object

As already stated before, methods are used to handle data contained inside an object. This is said to be instantiating an object. Clipper permits also to handle directly (apparently without methods) some variables contained inside objects. These are called ‘‘Exported Instance Variables’’. So, an object can be instantiated this way:


```
object : exported_instance_variable := new_value
```

```
object : method()
```

An exported instance variable may be read and/or modified depending on the allowed access to it; a method, inside Clipper, is something like a function with or without parameters (if parameters are present, these are usually used to modify data inside the object), that normally returns a value.

The ‘‘send’’ symbol

To instantiate an object or simply to access an exported instance variable, the ‘‘send’’ symbol (colon) is used.

More about objects

If this is not enough to understand objects inside Clipper, the following document should be read:

Peter M. Freese, *o:Clip - An Object Oriented Extension to Clipper 5.01* 1991, CyberSoft

ftp://ftp.simtel.net/pub/simtelnet/msdos/clipper/oclip.zip

Step 4: understand the get object

What happens with a command like the following:

```
@ nTop, nLeft GET Var
```

A get object is created containing all the necessary information for editing the variable ***Var*** at the screen position ***nTop***, ***nLeft***. After that, this get object is added to a get objects array (usually called ‘**GetList**’). The get objects array will contain all the get objects used during a ‘**READ**’.

So, what happens when a ‘**READ**’ command is encountered. The get objects array (‘**GetList**’) is read and the editing of all get objects is executed. After that, the get objects array is cleared.

This method hides what Clipper really makes. The suggestion here is to create a ‘**GET()**’ function that will substitute the ‘**@...GET**’ command and to use the ‘**READMODAL()**’ function to read the get objects array. Here is an example of it:

```
function GET( aoGet, nRow, nCol, bVar, cGetPicture,
              cColorString, bPreValid, bPostValid )

    // declare a local get object
    local oGet

    // create the get object using the function GETENV()
    oGet := GETENV( nRow, nCol, bVar, NIL, cGetPicture, cGetColor )

    // send to the get object the pre-validation condition code block (WHEN)
    oGet:preBlock  := bPreValid

    // send to the get object the post-validation condition code block (VALID)
    oGet:postBlock := bPostValid

    // display the get on the screen using the display() method
    oGet:display()

    // add the get object to the get objects array
    AADD( aoGet, oGet )

    return NIL
```

- ‘**aoGet**’ is the get objects array (so here is explicitly passed). This get objects array is modified (grown) and there is no need to return it as inside Clipper, arrays are always passed by reference to functions.
- ‘**nRow**’ and ‘**nCol**’ are the screen coordinates where the get field should appear at, as it works with the ‘**@...GET**’ command.
- ‘**bVar**’ is a special code block that permits the editing of a variable. If the variable ‘**Var**’ is to be edited, the code block is:

```
{ |x| iif( pcount() > 0, Var := x, Var }
```

- '**cGetPicture**' is the picture to use: same as the '**@...GET**' command.
- '**cColorString**' is the color string to use: same as the '**@...GET**' command.
- '**bPreValid**' is a code block containing the condition that must be valid before the cursor can reach this get field. It is equivalent to the '**WHEN**' condition used with the '**@...GET**' command, but it must be converted into a code block. For example, if the condition is '**A > B**', the code block is '**{|| A > B}**'
- '**bPostValid**' is a code block containing the condition that must be valid before the cursor can leave this get field. It is equivalent to the '**VALID**' condition used with the '**@...GET**' command, but it must be converted into a code block. For example, if the condition is '**A > B**', the code block is '**{|| A > B}**'

If there is a get function like the above one, screen I/O may be performed like the following example:

```
function do_some_editing()

   // define a variable to use as a get objects array
   // and initialise it to the empty array
   local aoGet := {}
   ...

   ...
   // add a new get object to the get objects array
   get(;
       aoGet,;
       10, 10,;
       { |x| iif( pcount() > 0, myVariable := x, myVariable },;
       "@s30@",;
       "gb+/b,  n/w,    n,  n,  w/n",;
       { || .T. },;
       { || .T. };
   )

   ...
   // read the get objects array
   readmodal( aoGet )

   // clear the get objects array
   aoGet := {}

   ...
   return ...
```

If the function '**GET()**' is not liked, the above I/O may be done as it follows:

```
function do_some_editing()

   // define a variable to use as a get object
   local aoGet

   // define a variable to use as a get objects array
   // and initialise it to the empty array
   local aoGet := {}
   ...

   ...
   // add a new get object to the get objects array

   oGet :=;
       GETENV(;
           10, 10,;
           { |x| iif( pcount() > 0, myVariable := x, myVariable },;
           NIL,;
           "@s30@",;
           "gb+/b,  n/w,    n,  n,  w/n",;
       )
   AADD( aoGet, oGet )

   ...
   // read the get objects array
   readmodal( aoGet )

   // clear the get objects array
   aoGet := {}

   ...
   return ...
```

Step 5: trying to stop using commands

To stop using commands, it is important to understand how commands are or may be translated into functions. Sometimes Clipper uses some undocumented functions: these are functions that start with a underline.

?/??

```
? [exp_list]
```

```
qout([exp_list])
```

```
?? [exp_list]
```

```
qqout([exp_list])
```

@...BOX

```
@ nTop, nLeft, nBottom, nRight BOX cnBoxString [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, [cnBoxString], [cColorString
])
```

@...GET

```
@ nTop, nLeft GET Var [PICTURE cGetPicture] [COLOR cColorString]
↩
↪[WHEN lPreExpression] [VALID lPostExpression]
```

```
setpos(nTop, nLeft)
```

```
aadd( GetList, _GET_( Var, "Var", cGetPicture, [
{|| lPostExpression}], ↩
↪[{|| lPreExpression}] ):display() ) atail(GetList):colorDisp(cColorSt
```

This is the command substitution made automatically, but it shouldn't be used to make clean programs. The step 4 (u0.1) suggests to create a get function.

@...SAY

```
@ nTop, nLeft SAY exp [COLOR cColorString]
```

```
devpos(nTop, nLeft)
```

```
devout(exp [, cColorString])
```

```
@ nTop, nLeft SAY exp PICTURE cSayPicture [COLOR cColorString]
```

```
devpos(nTop, nLeft)
```

```
devoutpic(exp, cSayPicture, [cColorString])
```

@...TO

```
@ nTop, nLeft TO nBottom, nRight DOUBLE [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, 2 [, cColorString])
```

```
@ nTop, nLeft TO nBottom, nRight [COLOR cColorString]
```

```
dispbox(nTop, nLeft, nBottom, nRight, 1 [,cColorString])
```

```
@ nTop, nLeft CLEAR [TO nBottom, nRight]
```

```
scroll([nTop], [nLeft], [nBottom, nRight])
```

```
setpos(nRow, nCol)
```

APPEND

```
APPEND BLANK
```

```
dbappend()
```

APPEND FROM

```
APPEND FROM xcFile [FIELDS idField_list] [scope] [WHILE lCondition
]↩
↪[FOR lCondition] [VIA xcDriver]
```

```
__dbApp( cFileName, [acFields], [bForCondition], [bWhileCondition]
, [nNextRecords],↩
↪[nRecord], [lRest], [cDriver] )
```

```
APPEND FROM xcFile [FIELDS idField_list] [scope] [WHILE lCondition
] [FOR lCondition]↩
↪DELIMITED xcDelimiter
```

```
__dbDelim( .f., cFileName, [cDelimiter], [acFields], [
bForCondition], ↩
↪[bWhileCondition], [nNextRecords], [nRecord], [lRest] )
```

```
APPEND FROM xcFile [FIELDS idField_list] [scope] [WHILE lCondition
]↩
↪[FOR lCondition] SDF
```

```
__dbSDF( .f., cFileName, [acFields], [bForCondition], [
bWhileCondition], ↩
↪[nNextRecords], [nRecord], [lRest] )
```

CLEAR

```
CLEAR
```

```
Scroll()
```

```
SetPos(0,0)
```

```
ReadKill(.T.)
```

```
GetList := {}
```

```
CLEAR GETS
```



```
ReadKill(.T.)
```

```
GetList := {}
```

```
CLEAR SCREEN | CLS
```

```
Scroll()
```

```
SetPos(0,0)
```

CLOSE

```
CLOSE
```

```
dbCloseArea()
```

```
CLOSE idAlias
```

```
idAlias->( dbCloseArea() )
```

```
CLOSE ALTERNATE
```

```
Set(19, "")
```

```
CLOSE DATABASES
```

```
dbCloseAll()
```

```
CLOSE INDEXES
```

```
dbClearIndex()
```

COMMIT

```
COMMIT
```

```
dbCommitAll()
```

CONTINUE

```
CONTINUE
```

```
__dbContinue()
```

COPY

```
COPY FILE xcSourceFile TO xcTargetFile | xcDevice
```

```
__CopyFile( cSourceFile, cTargetFile | cDevice )
```

```
COPY STRUCTURE [FIELDS idField_list] TO xcDatabase
```

```
__dbCopyStruct( cDatabase, [acFields] )
```

```
COPY STRUCTURE EXTENDED TO xcExtendedDatabase
```

```
__dbCopyXStruct( cExtendedDatabase )
```

```
COPY TO xcFile [FIELDS idField_list] [scope] [WHILE lCondition]↩
↪[FOR lCondition] [VIA xcDriver]
```

```
__dbCopy( cFileName, [acFields], [bForCondition], [bWhileCondition
], [nNextRecords],↩
↪[nRecord], [lRest], [cDriver] )
```

```
COPY TO xcFile [FIELDS idField_list] [scope] [WHILE lCondition] [
FOR lCondition]↩
↪DELIMITED xcDelimiter
```

```
__dbDelim( .t., cFileName, [cDelimiter], [acFields], [
bForCondition], ↩
↪[bWhileCondition], [nNextRecords], [nRecord], [lRest] )
```

```
COPY TO xcFile [FIELDS idField_list] [scope] [WHILE lCondition]↩
↪[FOR lCondition] SDF
```

```
__dbSDF( .t., cFileName, [acFields], [bForCondition], [
bWhileCondition], ↩
↪[nNextRecords], [nRecord], [lRest] )
```

COUNT

```
COUNT TO idVar [FOR lForCondition] [WHILE lWhileCondition] [
NEXT nNextRecords]↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||idVar:=idVar+1}, {||lForCondition}, {||lWhileCondition},↩
↪nNextRecords, nRecord, lRest )
```

CREATE

```
CREATE xcDatabase FROM xcExtendedDatabase [NEW] [ALIAS cAlias] [
VIA cDriver]
```

```
__dbCreate( cDatabase, cExtendedDatabase, [cDriver], [lNew], [
cAlias] )
```

DEFAULT

```
DEFAULT xVar TO xDefaultValue
```

```
if xVar == NIL
   xVar := xDefaultValue
end
```


DELETE

```
DELETE
```

```
dbDelete()
```

```
DELETE [FOR lForCondition] [WHILE lWhileCondition] [
NEXT nNextRecords]↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||dbDelete()}, {||lForCondition}, {||lWhileCondition},↩
↪nNextRecords, nRecord, lRest )
```

```
DELETE FILE xcFile
```

```
ferase( cFile )
```

EJECT

```
EJECT
```

```
qqout( chr(13) )
```

ERASE

```
ERASE xcFile
```

```
ferase( cFile )
```

FIND

```
FIND xcSearchString
```

```
dbSeek( cSearchString )
```

GO

```
GO[TO] nRecord
```

```
dbgoto(nRecord)
```

```
GO[TO] BOTTOM
```

```
dbGoBottom()
```

```
GO[TO] TOP
```

```
dbgotop()
```

INDEX ON

```
INDEX ON expKey TO xcIndexName [UNIQUE] [FOR lForCondition]↩
↪[WHILE lWhileCondition] [[EVAL lEvalCondition] [EVERY nRecords]
] ↩
↪[ASCENDING|DESCENDING]
```

```
ordCondSet( [cForCondition], [bForCondition], , [bWhileCondition]
,↩
↪[bEvalCondition], [nRecords], RECNO(), , , , lDescending )
```

```
ordCreate( cIndexName, , cExpKey, bExpKey, lUnique )
```

JOIN

```
JOIN WITH xcAlias TO xcDatabase [FOR lCondition] [
FIELDS idField_list]
```

```
__dbJoin( cAlias, cDatabase, [acFields], [bForCondition] )
```

KEYBOARD

```
KEYBOARD cString
```

```
__Keyboard( [cString] ) --> NIL
```

LABEL FORM

```
LABEL FORM xcLabel [TO PRINTER] [TO FILE xcFile] [NOCONSOLE]
[scope]↩
↪[WHILE lCondition] [FOR lCondition] [SAMPLE]
```

```
__LabelForm( cLabel, [lToPrinter], [cFile], [lNoConsole],↩
↪[bForCondition], [bWhileCondition], [nNextRecords], [nRecord],↩
↪[lRest], [lSample] )
```

LIST

```
LIST exp_list [TO PRINTER] [TO FILE xcFile] [scope]↩
↪[WHILE lCondition] [FOR lCondition] [OFF]
```

```
__dbList( [lToDisplay], abListColumns, [lAll], [bForCondition], [
bWhileCondition],↩
↪[nNextRecords], [nRecord], [lRest], [lToPrinter], [cFileName] )
```

LOCATE

```
LOCATE [scope] FOR lCondition [WHILE lCondition]
```

```
__dbLocate( [bForCondition], [bWhileCondition], [nNextRecords], [
nRecord], [lRest] )
```

PACK

```
PACK
```

```
__dbPack()
```

QUIT

```
QUIT
```

```
__Quit()
```


READ

```
READ
```

```
ReadModal(GetList)
```

```
GetList := {}
```

```
READ SAVE
```

```
ReadModal(GetList)
```

RECALL

```
RECALL
```

```
dbRecall()
```

```
RECALL [FOR lForCondition] [WHILE lWhileCondition] [
NEXT nNextRecords]↩
↪[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {||dbRecall()}, {||lForCondition}, {||lWhileCondition},↩
↪nNextRecords, nRecord, lRest )
```

REINDEX

```
REINDEX [EVAL lEvalCondition] [EVERY nRecords]
```

```
ordCondSet(, , , , [bEvalCondition], [nRecords], , , , , , , , )
```

```
ordListRebuild()
```

RELEASE

```
RELEASE idMemvar
```

```
__MXRelease( "idMemvar" )
```

```
RELEASE ALL
```

```
__MRelease("*", .t.)
```

```
RELEASE ALL LIKE skeleton
```

```
__MRelease( "skeleton", .t. )
```

```
RELEASE ALL EXCEPT skeleton
```

```
__MRelease( "skeleton", .F. )
```

RENAME

```
RENAME xcOldFile TO xcNewFile
```

```
frename( cOldFile, cNewFile )
```

REPLACE

```
REPLACE idField1 WITH exp1 [, idField2 WITH exp2...]
[FOR lForCondition] [WHILE lWhileCondition] [NEXT nNextRecords]
[RECORD nRecord] [REST] [ALL]
```

```
dbeval( {|| idField1 := exp1 [, idField2 := exp2...]}, {||lForCondition}, {||lWhileCondition}, nNextRecords,
nRecord, lRest )
```

```
REPLACE idField1 WITH exp1
```

```
idField1 := exp1
```

REPORT FORM

```
REPORT FORM xcReport [TO PRINTER] [TO FILE xcFile] [NOCONSOLE
] [scope]
[WHILE lCondition] [FOR lCondition] [PLAIN | HEADING cHeading]
[NOEJECT]
[SUMMARY]
```

```
__ReportForm( cForm, [lToPrinter], [cToFile], [lNoConsole], [
bForCondition], [bWhileCondition], [nNext], [nRecord], [lRest], [lPlain], [
cbHeading], [lBeforeEject], [lSummary] )
```

RESTORE

```
RESTORE SCREEN FROM cScreen
```

```
restscreen( 0, 0, Maxrow(), Maxcol(), cScreen )
```

RESTORE FROM

```
RESTORE FROM xcMemFile [ADDITIVE]
```

```
__MRestore( cMemFileName, [lAdditive] )
```

RUN

```
RUN xcCommandLine
```

```
__Run( cCommand )
```

SAVE SCREEN TO

```
SAVE SCREEN TO cScreen
```

```
cScreen := savescreen( 0, 0, maxrow(), maxcol() )
```


SAVE TO

```
SAVE TO xcMemFile [ALL [LIKE|EXCEPT skeleton]]
```

```
_MSave( cMemFileName, [cSkeleton], [lLike] )
```

SEEK

```
SEEK expSearch [SOFTSEEK]
```

```
dbSeek( expSearch [, lSoftSeek] )
```

SELECT

```
SELECT xnWorkArea | idAlias
```

```
dbSelectArea( nWorkArea | cIdAlias )
```

SET

Most of the '**SET**...' commands are translated into the '**SET()**' function that distinguishes different modes depending on a number. As this number is difficult to handle during programming (essentially because it is difficult to remember the meaning of it), Clipper offers the '**SET.CH**' include file that helps with manifest constants.

```
#define _SET_EXACT         1
#define _SET_FIXED         2
#define _SET_DECIMALS      3
#define _SET_DATEFORMAT    4
#define _SET_EPOCH         5
#define _SET_PATH          6
#define _SET_DEFAULT       7

#define _SET_EXCLUSIVE     8
#define _SET_SOFTSEEK      9
#define _SET_UNIQUE       10
#define _SET_DELETED      11

#define _SET_CANCEL       12
#define _SET_DEBUG        13
#define _SET_TYPEAHEAD    14

#define _SET_COLOR        15
#define _SET_CURSOR       16
#define _SET_CONSOLE      17
#define _SET_ALTERNATE    18
#define _SET_ALTFILE      19
#define _SET_DEVICE       20
#define _SET_EXTRA        21
#define _SET_EXTRAFILE    22
#define _SET_PRINTER      23
#define _SET_PRINTFILE    24
#define _SET_MARGIN       25

#define _SET_BELL         26
#define _SET_CONFIRM      27
#define _SET_ESCAPE       28
#define _SET_INSERT       29
#define _SET_EXIT         30
#define _SET_INTENSITY    31
#define _SET_SCOREBOARD   32
#define _SET_DELIMITERS   33
#define _SET_DELIMCHARS   34

#define _SET_WRAP         35
#define _SET_MESSAGE      36
#define _SET_MCENTER      37
#define _SET_SCROLLBREAK  38
```

```
SET ALTERNATE TO xcFile [ADDITIVE]
```

```
Set( _SET_ALTFILE, cFile, lAdditive )
```

```
SET ALTERNATE ON | OFF | xlToggle
```

```
Set( _SET_ALTERNATE, "ON" | "OFF" | lToggle )
```

```
SET BELL ON | OFF | xlToggle
```

```
Set( _SET_BELL, "ON" | "OFF" | lToggle )
```

```
SET COLOR | COLOUR TO (cColorString)
```

```
SetColor( cColorString )
```

```
SET CONFIRM ON | OFF | xlToggle
```

```
Set( _SET_CONFIRM, "ON" | "OFF" | lToggle )
```

```
SET CONSOLE ON | OFF | xlToggle
```

```
Set( _SET_CONSOLE, "ON" | "OFF" | lToggle )
```

```
SET CURSOR ON | OFF | xlToggle
```

```
SetCursor( 1 | 0 | iif( lToggle, 1, 0 ) )
```

```
SET DATE FORMAT [TO] cDateFormat
```

```
Set( _SET_DATEFORMAT, cDateFormat )
```

```
SET DECIMALS TO
```

```
Set( _SET_DECIMALS, 0 )
```

```
SET DECIMALS TO nDecimals
```

```
Set( _SET_DECIMALS, nDecimals )
```

```
SET DEFAULT TO
```

```
Set( _SET_DEFAULT, "" )
```

```
SET DEFAULT TO xcPathspec
```

```
Set( _SET_DEFAULT, cPathspec )
```

```
SET DELETED ON | OFF | xlToggle
```

```
Set( _SET_DELETED, "ON" | "OFF" | lToggle )
```

```
SET DELIMITERS ON | OFF | xlToggle
```



```
Set( _SET_DELIMITERS, "ON" | "OFF" | lToggle )
```

```
SET DELIMITERS TO [DEFAULT]
```

```
Set( _SET_DELIMCHARS, "::" )
```

```
SET DELIMITERS TO cDelimiters
```

```
Set( _SET_DELIMCHARS, cDelimiters )
```

```
SET DEVICE TO SCREEN | PRINTER
```

```
Set( _SET_DEVICE, "SCREEN" | "PRINTER" )
```

```
SET EPOCH TO nYear
```

```
Set( _SET_EPOCH, nYear )
```

```
SET ESCAPE ON | OFF | xlToggle
```

```
Set( _SET_ESCAPE, "ON" | "OFF" | lToggle )
```

```
SET EXACT ON | OFF | xlToggle
```

```
Set( _SET_EXACT, "ON" | "OFF" | lToggle )
```

```
SET EXCLUSIVE ON | OFF | xlToggle
```

```
Set( _SET_EXCLUSIVE, "ON" | "OFF" | lToggle )
```

```
SET FILTER TO
```

```
dbclearfilter()
```

```
SET FILTER TO lCondition
```

```
dbsetfilter( bCondition, cCondition )
```

```
SET FIXED ON | OFF | xlToggle
```

```
Set( _SET_FIXED, "ON" | "OFF" | lToggle )
```

```
SET FUNCTION nFunctionKey TO cString
```

```
__SetFunction( nFunctionKey, cString )
```

```
SET INDEX TO [xcIndex [, xcIndex1...] ]
```

```
ordListClear()
```

```
ordListAdd( cIndex )
ordListAdd( cIndex1 )
…
```

```
SET INTENSITY ON | OFF | xlToggle
```

```
Set( _SET_INTENSITY, "ON" | "OFF" | lToggle )
```

```
SET KEY nInkeyCode [TO]
```

```
SetKey( nInkeyCode, NIL )
```

```
SET KEY nInkeyCode TO [idProcedure]
```

```
SetKey( nInkeyCode, { |p, l, v| idProcedure(p, l, v)} )
```

```
SET MARGIN TO
```

```
Set( _SET_MARGIN, 0 )
```

```
SET MARGIN TO [nPageOffset]
```

```
Set( _SET_MARGIN, nPageOffset )
```

```
SET MESSAGE TO
```

```
Set( _SET_MESSAGE, 0 )
```

```
Set( _SET_MCENTER, .F. )
```

```
SET MESSAGE TO [nRow [CENTER | CENTRE]]
```

```
Set( _SET_MESSAGE, nRow )
```

```
Set( _SET_MCENTER, lCenter )
```

```
SET ORDER TO [nIndex]
```

```
ordSetFocus( nIndex )
```

```
SET PATH TO
```

```
Set( _SET_PATH, "" )
```

```
SET PATH TO [xcPathspec [, cPathspec1… ] ]
```



```
Set( _SET_PATH, cPathspec [, cPathspec1… ] )
```

```
SET PRINTER ON | OFF | xlToggle
```

```
Set( _SET_PRINTER, "ON" | "OFF" | lToggle )
```

```
SET PRINTER TO
```

```
Set( _SET_PRINTFILE, "" )
```

```
SET PRINTER TO [xcDevice | xcFile [ADDITIVE]]
```

```
Set( _SET_PRINTFILE, cDevice | cFile, lAdditive )
```

```
SET RELATION TO
```

```
dbclearrelation()
```

```
SET RELATION TO [expKey1 INTO xcAlias1] [, [TO]
expKey2 INTO xcAlias2…]
```

```
dbClearRel()
```

```
dbSetRelation( cAlias1, {||expKey1}, ["expKey1"] )
```

```
dbSetRelation( cAlias2, {||expKey2}, ["expKey1"] )
```

```
SET RELATION TO [expKey1 INTO xcAlias1]↩
↪[, [TO] expKey2 INTO xcAlias2…] ADDITIVE
```

```
dbSetRelation( cAlias1, {||expKey1}, ["expKey1"] )
```

```
dbSetRelation( cAlias2, {||expKey2}, ["expKey1"] )
```

```
SET SCOREBOARD ON | OFF | xlToggle
```

```
Set( _SET_SCOREBOARD, "ON" | "OFF" | lToggle )
```

```
SET SOFTSEEK ON | OFF | xlToggle
```

```
Set( _SET_SOFTSEEK, "ON" | "OFF" | lToggle )
```

```
SET TYPEAHEAD TO nKeyboardSise
```

```
Set( _SET_TYPEAHEAD, nKeyboardSise )
```

```
SET UNIQUE ON | OFF | xlToggle
```

```
Set( _SET_UNIQUE, "ON" | "OFF" | lToggle )
```

```
SET WRAP ON | OFF | xlToggle
```

```
Set( _SET_WRAP, "ON" | "OFF" | lToggle )
```

SKIP

```
SKIP [nRecords] [ALIAS idAlias|nWorkArea]
```

```
[idAlias|nWorkArea -> ]( dbSkip([nRecords]) )
```

SORT

```
SORT TO xcDatabase ON idField1 [/[A|D][C]] [, idField2 [/[A|D][C]] ...]
↪[scope] [WHILE lCondition] [FOR lCondition]
```

```
__dbSort( cDatabase, [acFields], [bForCondition], [bWhileCondition],
↪[nNextRecords], [nRecord], [lRest] )
```

STORE

```
STORE value TO variable
```

```
variable := value
```

SUM

```
SUM nExp1 [, nExp2...] TO idVar1 [, idVar2...] [FOR lForCondition]
↪[WHILE lWhileCondition] [NEXT nNextRecords] [RECORD nRecord] [
REST] [ALL]
```

```
dbeval( {||idVar1:=idVar1+nExp1 [, idVar2:=idVar2+nExp2...] },
↪{||lForCondition}, {||lWhileCondition}, nNextRecords, nRecord, lRest )
```

TOTAL ON

```
TOTAL ON expKey [FIELDS idField_list] TO xcDatabase [scope]
↪[WHILE lCondition] [FOR lCondition]
```

```
__dbTotal( cDatabase, bKey, [acFields], [bForCondition], [
bWhileCondition],
↪[nNextRecords], [nRecord], [lRest] )
```

UNLOCK

```
UNLOCK
```

```
dbUnlock()
```

```
UNLOCK ALL
```

```
dbUnlockAll()
```


UPDATE FROM

```
UPDATE FROM xcAlias ON expKey [RANDOM] REPLACE idField1
↪WITH exp [, idField2 WITH exp ...]
```

```
__dbUpdate( cAlias, bKey, [lRandom], [bReplacement] )
```

Example:

```
__dbUpdate( "INVOICE", {|| LAST}, .T.,; {|| FIELD->TOTAL1 :=
↪INVOICE->SUM1,; FIELD->TOTAL2 := INVOICE->SUM2 } )
```

USE

```
USE
```

```
dbclosearea()
```

```
USE [xcDatabase]
↪[INDEX xcIndex1 [, xcIndex2...] [ALIAS xcAlias] [EXCLUSIVE|
SHARED] [NEW]
↪[READONLY] [VIA cDriver]]
```

```
dbUseArea( [lNewArea], [cDriver], cDatabase, [cAlias], [lShared]
, [lReadOnly] )
```

```
[dbSetIndex( cIndex1 )]
```

```
[dbSetIndex( cIndex2 )]
```

```
...
```

ZAP

```
ZAP
```

```
dbZap()
```

Step 6: free yourself from STD.CH - /U

Now that no command is used, the standard include file 'STD.CH' is no more necessary. Clipper uses 'STD.CH' automatically, unless specified differently. Just compile this way:

C:>**CLIPPER TEST.PRG /U** [*Enter*]

Step 7: take control over all include files

Clipper comes with so many include files ('*.CH'). To avoid confusion, a single 'STANDARD.CH' file containing all what is needed for the application may be prepared. At least, it is necessary the following.

```
+================================================================
* DISPBOX()
+================================================================

* Single-line box
#define BOX_SINGLE;
    (;
        CHR(218) +;
        CHR(196) +;
        CHR(191) +;
        CHR(179) +;
        CHR(217) +;
        CHR(196) +;
        CHR(192) +;
        CHR(179);
    )
```

```
* Double-line box
#define BOX_DOUBLE;
    (;
        CHR(201) +;
        CHR(205) +;
        CHR(187) +;
        CHR(186) +;
        CHR(188) +;
        CHR(205) +;
        CHR(200) +;
        CHR(186);
    )

* Single-line top, double-line sides
#define BOX_SINGLE_DOUBLE;
    (;
        CHR(214) +;
        CHR(196) +;
        CHR(183) +;
        CHR(186) +;
        CHR(189) +;
        CHR(196) +;
        CHR(211) +;
        CHR(186);
    )

* Double-line top, single-line sides
#define BOX_DOUBLE_SINGLE;
    (;
        CHR(213) +;
        CHR(205) +;
        CHR(184) +;
        CHR(179) +;
        CHR(190) +;
        CHR(205) +;
        CHR(212) +;
        CHR(179);
    )


*==============================================================
* ERRORS
*==============================================================

* Severity levels (e:severity)
#define ERROR_SEVERITY_WHOCARES        0
#define ERROR_SEVERITY_WARNING         1
#define ERROR_SEVERITY_ERROR           2
#define ERROR_SEVERITY_CATASTROPHIC    3

* Generic error codes (e:genCode)
#define ERROR_GENERIC_ARG              1
#define ERROR_GENERIC_BOUND            2
#define ERROR_GENERIC_STROVERFLOW      3
#define ERROR_GENERIC_NUMOVERFLOW      4
#define ERROR_GENERIC_ZERODIV          5
#define ERROR_GENERIC_NUMERR           6
#define ERROR_GENERIC_SYNTAX           7
#define ERROR_GENERIC_COMPLEXITY       8

#define ERROR_GENERIC_MEM             11
#define ERROR_GENERIC_NOFUNC          12
#define ERROR_GENERIC_NOMETHOD        13
#define ERROR_GENERIC_NOVAR           14
#define ERROR_GENERIC_NOALIAS         15
#define ERROR_GENERIC_NOVARMETHOD     16
#define ERROR_GENERIC_BADALIAS        17
#define ERROR_GENERIC_DUPALIAS        18

#define ERROR_GENERIC_CREATE          20
#define ERROR_GENERIC_OPEN            21
#define ERROR_GENERIC_CLOSE           22
#define ERROR_GENERIC_READ            23
#define ERROR_GENERIC_WRITE           24
#define ERROR_GENERIC_PRINT           25

#define ERROR_GENERIC_UNSUPPORTED     30
#define ERROR_GENERIC_LIMIT           31
#define ERROR_GENERIC_CORRUPTION      32
#define ERROR_GENERIC_DATATYPE        33
#define ERROR_GENERIC_DATAWIDTH       34
#define ERROR_GENERIC_NOTABLE         35
#define ERROR_GENERIC_NOORDER         36
#define ERROR_GENERIC_SHARED          37
#define ERROR_GENERIC_UNLOCKED        38
#define ERROR_GENERIC_READONLY        39

#define ERROR_GENERIC_APPENDLOCK      40
#define ERROR_GENERIC_LOCK            41


*==============================================================
* INKEY()
*==============================================================

#define K_UP                5   //   Up arrow, Ctrl-E
#define K_DOWN             24   //   Down arrow, Ctrl-X
#define K_LEFT             19   //   Left arrow, Ctrl-S
#define K_RIGHT             4   //   Right arrow, Ctrl-D
#define K_HOME              1   //   Home, Ctrl-A
#define K_END               6   //   End, Ctrl-F
#define K_PGUP             18   //   PgUp, Ctrl-R
#define K_PGDN              3   //   PgDn, Ctrl-C
```



```
#define K_CTRL_UP         397   // * Ctrl-Up arrow
#define K_CTRL_DOWN       401   // * Ctrl-Down arrow
#define K_CTRL_LEFT        26   //   Ctrl-Left arrow, Ctrl-Z
#define K_CTRL_RIGHT        2   //   Ctrl-Right arrow, Ctrl-B
#define K_CTRL_HOME        29   //   Ctrl-Home, Ctrl-</synsqb>
#define K_CTRL_END         23   //   Ctrl-End, Ctrl-W
#define K_CTRL_PGUP        31   //   Ctrl-PgUp, Ctrl-Hyphen
#define K_CTRL_PGDN        30   //   Ctrl-PgDn, Ctrl-^

#define K_ALT_UP          408   // * Alt-Up arrow
#define K_ALT_DOWN        416   // * Alt-Down arrow
#define K_ALT_LEFT        411   // * Alt-Left arrow
#define K_ALT_RIGHT       413   // * Alt-Right arrow
#define K_ALT_HOME        407   // * Alt-Home
#define K_ALT_END         415   // * Alt-End
#define K_ALT_PGUP        409   // * Alt-PgUp
#define K_ALT_PGDN        417   // * Alt-PgDn

#define K_ENTER            13   //   Enter, Ctrl-M
#define K_RETURN           13   //   Return, Ctrl-M
#define K_SPACE            32   //   Space bar
#define K_ESC              27   //   Esc, Ctrl-<synsqb>

#define K_CTRL_ENTER       10   //   Ctrl-Enter
#define K_CTRL_RETURN      10   //   Ctrl-Return
#define K_CTRL_RET         10   //   Ctrl-Return (Compat.)
#define K_CTRL_PRTSCR     379   // * Ctrl-Print Screen
#define K_CTRL_QUESTION   309   //   Ctrl-?

#define K_ALT_ENTER       284   // * Alt-Enter
#define K_ALT_RETURN      284   // * Alt-Return
#define K_ALT_EQUALS      387   // * Alt-Equals
#define K_ALT_ESC         257   // * Alt-Esc

#define KP_ALT_ENTER      422   // * Keypad Alt-Enter

#define KP_CTRL_5         399   // * Keypad Ctrl-5
#define KP_CTRL_SLASH     405   // * Keypad Ctrl-/
#define KP_CTRL_ASTERISK  406   // * Keypad Ctrl-*
#define KP_CTRL_MINUS     398   // * Keypad Ctrl--
#define KP_CTRL_PLUS      400   // * Keypad Ctrl-+

#define KP_ALT_5            5   // * Keypad Alt-5
#define KP_ALT_SLASH      420   // * Keypad Alt-/
#define KP_ALT_ASTERISK   311   // * Keypad Alt-*
#define KP_ALT_MINUS      330   // * Keypad Alt--
#define KP_ALT_PLUS       334   // * Keypad Alt-+

#define K_INS              22   //   Ins, Ctrl-V
#define K_DEL               7   //   Del, Ctrl-G
#define K_BS                8   //   Backspace, Ctrl-H
#define K_TAB               9   //   Tab, Ctrl-I
#define K_SH_TAB          271   //   Shift-Tab

#define K_CTRL_INS        402   // * Ctrl-Ins
#define K_CTRL_DEL        403   // * Ctrl-Del
#define K_CTRL_BS         127   //   Ctrl-Backspace
#define K_CTRL_TAB        404   // * Ctrl-Tab

#define K_ALT_INS         418   // * Alt-Ins
#define K_ALT_DEL         419   // * Alt-Del
#define K_ALT_BS          270   // * Alt-Backspace
#define K_ALT_TAB         421   // * Alt-Tab

#define K_CTRL_A            1   //   Ctrl-A, Home
#define K_CTRL_B            2   //   Ctrl-B, Ctrl-Right arrow
#define K_CTRL_C            3   //   Ctrl-C, PgDn, Ctrl-ScrollLock
#define K_CTRL_D            4   //   Ctrl-D, Right arrow
#define K_CTRL_E            5   //   Ctrl-E, Up arrow
#define K_CTRL_F            6   //   Ctrl-F, End
#define K_CTRL_G            7   //   Ctrl-G, Del
#define K_CTRL_H            8   //   Ctrl-H, Backspace
#define K_CTRL_I            9   //   Ctrl-I, Tab
#define K_CTRL_J           10   //   Ctrl-J
#define K_CTRL_K           11   //   Ctrl-K
#define K_CTRL_L           12   //   Ctrl-L
#define K_CTRL_M           13   //   Ctrl-M, Return
#define K_CTRL_N           14   //   Ctrl-N
#define K_CTRL_O           15   //   Ctrl-O
#define K_CTRL_P           16   //   Ctrl-P
#define K_CTRL_Q           17   //   Ctrl-Q
#define K_CTRL_R           18   //   Ctrl-R, PgUp
#define K_CTRL_S           19   //   Ctrl-S, Left arrow
#define K_CTRL_T           20   //   Ctrl-T
#define K_CTRL_U           21   //   Ctrl-U
#define K_CTRL_V           22   //   Ctrl-V, Ins
#define K_CTRL_W           23   //   Ctrl-W, Ctrl-End
#define K_CTRL_X           24   //   Ctrl-X, Down arrow
#define K_CTRL_Y           25   //   Ctrl-Y
#define K_CTRL_Z           26   //   Ctrl-Z, Ctrl-Left arrow

#define K_ALT_A           286   //   Alt-A
#define K_ALT_B           304   //   Alt-B
#define K_ALT_C           302   //   Alt-C
#define K_ALT_D           288   //   Alt-D
#define K_ALT_E           274   //   Alt-E
#define K_ALT_F           289   //   Alt-F
#define K_ALT_G           290   //   Alt-G
#define K_ALT_H           291   //   Alt-H
#define K_ALT_I           279   //   Alt-I
#define K_ALT_J           292   //   Alt-J
#define K_ALT_K           293   //   Alt-K
```

```
#define K_ALT_L          294   //   Alt-L
#define K_ALT_M          306   //   Alt-M
#define K_ALT_N          305   //   Alt-N
#define K_ALT_O          280   //   Alt-O
#define K_ALT_P          281   //   Alt-P
#define K_ALT_Q          272   //   Alt-Q
#define K_ALT_R          275   //   Alt-R
#define K_ALT_S          287   //   Alt-S
#define K_ALT_T          276   //   Alt-T
#define K_ALT_U          278   //   Alt-U
#define K_ALT_V          303   //   Alt-V
#define K_ALT_W          273   //   Alt-W
#define K_ALT_X          301   //   Alt-X
#define K_ALT_Y          277   //   Alt-Y
#define K_ALT_Z          300   //   Alt-Z
#define K_ALT_1          376   //   Alt-1
#define K_ALT_2          377   //   Alt-2
#define K_ALT_3          378   //   Alt-3
#define K_ALT_4          379   //   Alt-4
#define K_ALT_5          380   //   Alt-5
#define K_ALT_6          381   //   Alt-6
#define K_ALT_7          382   //   Alt-7
#define K_ALT_8          383   //   Alt-8
#define K_ALT_9          384   //   Alt-9
#define K_ALT_0          385   //   Alt-0

#define K_F1              28   //   F1, Ctrl-Backslash
#define K_F2              -1   //   F2
#define K_F3              -2   //   F3
#define K_F4              -3   //   F4
#define K_F5              -4   //   F5
#define K_F6              -5   //   F6
#define K_F7              -6   //   F7
#define K_F8              -7   //   F8
#define K_F9              -8   //   F9
#define K_F10             -9   //   F10
#define K_F11            -40   // * F11
#define K_F12            -41   // * F12

#define K_CTRL_F1        -20   //   Ctrl-F1
#define K_CTRL_F2        -21   //   Ctrl-F2
#define K_CTRL_F3        -22   //   Ctrl-F4
#define K_CTRL_F4        -23   //   Ctrl-F3
#define K_CTRL_F5        -24   //   Ctrl-F5
#define K_CTRL_F6        -25   //   Ctrl-F6
#define K_CTRL_F7        -26   //   Ctrl-F7
#define K_CTRL_F8        -27   //   Ctrl-F8
#define K_CTRL_F9        -28   //   Ctrl-F9
#define K_CTRL_F10       -29   //   Ctrl-F10
#define K_CTRL_F11       -44   // * Ctrl-F11
#define K_CTRL_F12       -45   // * Ctrl-F12

#define K_ALT_F1         -30   //   Alt-F1
#define K_ALT_F2         -31   //   Alt-F2
#define K_ALT_F3         -32   //   Alt-F3
#define K_ALT_F4         -33   //   Alt-F4
#define K_ALT_F5         -34   //   Alt-F5
#define K_ALT_F6         -35   //   Alt-F6
#define K_ALT_F7         -36   //   Alt-F7
#define K_ALT_F8         -37   //   Alt-F8
#define K_ALT_F9         -38   //   Alt-F9
#define K_ALT_F10        -39   //   Alt-F10
#define K_ALT_F11        -46   // * Alt-F11
#define K_ALT_F12        -47   // * Alt-F12

#define K_SH_F1          -10   //   Shift-F1
#define K_SH_F2          -11   //   Shift-F2
#define K_SH_F3          -12   //   Shift-F3
#define K_SH_F4          -13   //   Shift-F4
#define K_SH_F5          -14   //   Shift-F5
#define K_SH_F6          -15   //   Shift-F6
#define K_SH_F7          -16   //   Shift-F7
#define K_SH_F8          -17   //   Shift-F8
#define K_SH_F9          -18   //   Shift-F9
#define K_SH_F10         -19   //   Shift-F10
#define K_SH_F11         -42   // * Shift-F11
#define K_SH_F12         -43   // * Shift-F12


*====================================================================
* MEMOEDIT()
*====================================================================

* User function entry modes
#define MEMOEDIT_IDLE           0      // idle, all keys processed
#define MEMOEDIT_UNKEY          1      // unknown key, memo unaltered
#define MEMOEDIT_UNKEYX         2      // unknown key, memo altered
#define MEMOEDIT_INIT           3      // initialization mode


* User function return codes
#define MEMOEDIT_DEFAULT        0      // perform default action
#define MEMOEDIT_IGNORE        32      // ignore unknown key
#define MEMOEDIT_DATA          33      // treat unknown key as data
#define MEMOEDIT_TOGGLEWRAP    34      // toggle word-wrap mode
#define MEMOEDIT_TOGGLESCROLL  35      // toggle scrolling mode
#define MEMOEDIT_WORDRIGHT    100      // perform word-right operation
#define MEMOEDIT_BOTTOMRIGHT  101      // perform bottom-right operation


*====================================================================
* SET()
*====================================================================
```



```
#define _SET_EXACT         1
#define _SET_FIXED         2
#define _SET_DECIMALS      3
#define _SET_DATEFORMAT    4
#define _SET_EPOCH         5
#define _SET_PATH          6
#define _SET_DEFAULT       7

#define _SET_EXCLUSIVE     8
#define _SET_SOFTSEEK      9
#define _SET_UNIQUE       10
#define _SET_DELETED      11

#define _SET_CANCEL       12
#define _SET_DEBUG        13
#define _SET_TYPEAHEAD    14

#define _SET_COLOR        15
#define _SET_CURSOR       16
#define _SET_CONSOLE      17
#define _SET_ALTERNATE    18
#define _SET_ALTFILE      19
#define _SET_DEVICE       20
#define _SET_EXTRA        21
#define _SET_EXTRAFILE    22
#define _SET_PRINTER      23
#define _SET_PRINTFILE    24
#define _SET_MARGIN       25

#define _SET_BELL         26
#define _SET_CONFIRM      27
#define _SET_ESCAPE       28
#define _SET_INSERT       29
#define _SET_EXIT         30
#define _SET_INTENSITY    31
#define _SET_SCOREBOARD   32
#define _SET_DELIMITERS   33
#define _SET_DELIMCHARS   34

#define _SET_WRAP         35
#define _SET_MESSAGE      36
#define _SET_MCENTER      37
#define _SET_SCROLLBREAK  38

*====================================================================
* SETCURSOR()
*====================================================================

#define SETCURSOR_NONE     0  // No cursor
#define SETCURSOR_NORMAL   1  // Normal cursor (underline)
#define SETCURSOR_INSERT   2  // Insert cursor (lower half block)
#define SETCURSOR_SPECIAL1 3  // Special cursor (full block)
#define SETCURSOR_SPECIAL2 4  // Special cursor (upper half block)


*====================================================================
* RDD REQUESTs
*====================================================================

external dbfndx
external dbfntx // default
```

[1] This material appeared originally at '`http://www.geocities.com/SiliconValley/7737/clipper52clean.html`', in 1996.

[2] **Clipper 5.2** Proprietary software

Parte xxxiii

Programmare a 16 bit

Microprocessori x86-16 1285

Segmenti 1285
Registri 1286
Trasferimento di dati tra due segmenti differenti 1288
Riferimenti a indirizzi di memoria con i registri 1288
Convenzioni di chiamata 1289
Sintesi delle istruzioni x86-16 1290
Sostituzione delle istruzioni per i186 1305
Riferimenti 1306

Architettura IBM PC 1307

IVT: «interrupt vector table» 1307
BIOS data area 1308
Altre aree di memoria 1309
Interruzioni principali 1309
Riferimenti 1311

Strumenti di sviluppo e di utilizzo 1313

Preparazione 1313
Bcc 1314
As86 1315
Ld86 1315
Bootblocks 1316
Riferimenti 1317

Segmenti 1285
Registri 1286
Trasferimento di dati tra due segmenti differenti 1288
Riferimenti a indirizzi di memoria con i registri 1288
Convenzioni di chiamata 1289
Sintesi delle istruzioni x86-16 1290
Sostituzione delle istruzioni per i186 1305
Riferimenti 1306

ADC 1296 ADD 1296 AH 1286 AL 1286 AND 1297 AX 1286 BH 1286 BL 1286 BP 1286 BX 1286 CALL 1298 CALL FAR 1298 CBW 1290 CH 1286 CL 1286 CLC 1301 CLD 1301 CLI 1301 CMC 1301 CMP 1301 CMPSB 1293 CMPSW 1293 CWD 1290 CX 1286 DEC 1296 DH 1286 DI 1286 DIV 1296 DL 1286 DX 1286 ENTER 1298 FLAGS 1286 HLT 1301 IDIV 1296 IMUL 1296 IN 1305 1305 INC 1296 INT 1301 INTO 1301 IP 1286 IRET 1301 JA 1302 JAE 1302 JB 1302 JBE 1302 JC 1302 JCXZ 1302 JE 1302 JG 1302 JGE 1302 JL 1302 JLE 1302 JMP 1302 JMP FAR 1302 JNA 1302 JNAE 1302 JNB 1302 JNBE 1302 JNC 1302 JNE 1302 JNG 1302 JNGE 1302 JNL 1302 JNO 1302 JNP 1302 JNS 1302 JNZ 1302 JO 1302 JP 1302 JPE 1302 JPO 1302 JS 1302 JZ 1302 LAHF 1290 LDS 1290 LEA 1290 LEAVE 1298 LES 1290 LODSB 1292 LODSW 1292 LOOP 1305 LOOPE 1305 LOOPNE 1305 LOOPNZ 1305 LOOPZ 1305 MOV 1290 MOVSB 1292 MOVSW 1292 MUL 1296 NEG 1296 NOP 1290 NOT 1297 OR 1297 OUT 1305 1305 POP 1298 POPA 1298 POPF 1298 PUSH 1298 PUSHA 1298 PUSHF 1298 RCL 1297 RCR 1297 REP 1292 REPE 1293 REPNE 1293 REPNZ 1293 REPZ 1293 RET 1298 RETF 1298 RET FAR 1298 ROL 1297 ROR 1297 SAHF 1290 SAL 1297 SAR 1297 SBB 1296 SCASB 1293 SCASW 1293 SHL 1297 SHR 1297 SI 1286 SP 1286 STC 1301 1301 STI 1301 STOSB 1292 STOSW 1292 SUB 1296 TEST 1301 XCHG 1290 XLATB 1292 XOR 1297

I microprocessori x86-16 sono sostanzialmente costituiti dal 8086 e dal 8088, con la caratteristica di gestire registri a 16 bit e di poter indirizzare complessivamente fino a 1024 Kibyte, suddividendo però la memoria in segmenti da 64 Kibyte. Questa famiglia ha il limite di disporre di pochi registri per usi generali, spesso vincolati a un ruolo preciso, nell'ambito di certe istruzioni.

Dal momento che esiste una grande quantità di modelli di microprocessori compatibili con la vecchia famiglia a 16 bit e dato che sono disponibili simulatori ed emulatori, può essere ancora interessante lo studio della programmazione a 16 bit, riferita al modello x86-16, se non si devono affrontare problematiche relative alla protezione della memoria e a gestioni sofisticate della stessa.

Questo e gli altri capitoli dedicati alla programmazione con i microprocessori x86-16 e l'architettura dell'elaboratore IBM PC dei primi anni 1980, si limitano ad affrontare le questioni che consentono di lavorare con i registri di segmento posti tutti allo stesso valore (salva la possibilità di travasare dei dati da una parte della memoria all'altra). Molte questioni importanti non vengono affrontate e si rimanda ai riferimenti posti alla fine dei capitoli, per gli approfondimenti eventuali, oltre che al capitolo 64, in cui si fa riferimento ai microprocessori x86-32.

Segmenti

Prima di considerare i registri di un microprocessore x86-16, è importante comprendere il concetto di ***segmento***, utilizzato in questo contesto.

Dal momento che i registri sono a 16 bit, con questi si possono rappresentare valori senza segno da zero a 65535; pertanto, dato che la memoria è organizzata in byte, con un registro si può scandire soltanto un intervallo di 64 Kibyte. Per poter scandire lo spazio di

1024 Kibyte, occorrono due registri, in modo da comporre assieme un indirizzo da 20 bit.

Per indirizzare la memoria, a qualunque titolo, nei microprocessori x86-16 è necessario un ***registro di segmento*** e un altro valore che esprima lo scostamento dall'inizio del segmento a cui il contesto si riferisce.

I segmenti possono collocarsi in memoria con una certa libertà, pertanto possono sovrapporsi, parzialmente o completamente. In pratica la memoria viene suddivisa idealmente in ***paragrafi*** (o *click*) da 16 byte ciascuno e i segmenti possono iniziare soltanto all'inizio di un paragrafo. Per garantire che ciò avvenga in questo modo, i registri che sono dedicati a rappresentare l'inizio di un segmento, riportano il numero del paragrafo, ovvero l'indirizzo assoluto di memoria diviso per 16. Questa divisione si ottiene con un semplice scorrimento a destra di quattro bit; pertanto, per ritrovare il valore originale è sufficiente fare lo scorrimento opposto, verso sinistra. Per esempio, il valore $159D_{16}$ contenuto in un registro di segmento, individua in realtà l'indirizzo $159D0_{16}$, pari a 88528_{10}.

Come accennato, per individuare una certa posizione in memoria si usa sempre un registro di segmento e un altro valore che rappresenta lo scostamento a partire dall'inizio del segmento a cui si riferisce il contesto. Per esempio, se il registro di segmento contiene il valore $159D_{16}$ e si specifica lo scostamento $FFFE_{16}$, si sta in pratica facendo riferimento alla posizione di memoria $159D0_{16}$+$FFFE_{16}$, pari a $259CE_{16}$, ovvero 154062_{10}.

Stante questa organizzazione, per indicare in un documento un certo indirizzo di memoria, si può usare la definizione di «indirizzo efficace» e si può scrivere un solo numero, come per esempio $159DF_{16}$.

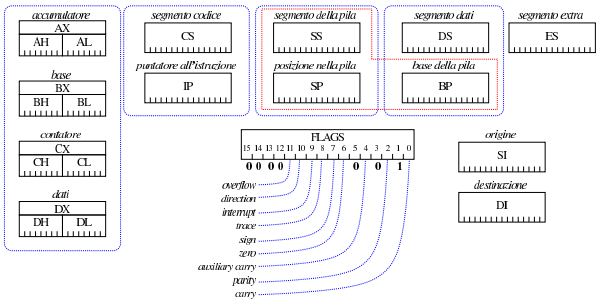
Riquadro u139.1. Valori affiancati e divisi dal simbolo '**:**'.

Nella programmazione a 16 bit, con i microprocessori della famiglia x86, per affiancare due valori a 16 bit si usa normalmente il segno di due punti, come per esempio ***DX*:*AX*** o $159D_{16}$:$FFFE_{16}$.
In generale, questa rappresentazione indica soltanto che si vuole fare riferimento a un numero a 32 bit, formato dall'unione delle due parti indicate, ma il significato che questo numero deve avere va interpretato in base al contesto. Per esempio, ***DX*:*AX*** potrebbe essere il risultato di una moltiplicazione, da prendere numericamente tale e quale, nel senso che il registro ***DX*** rappresenta i 16 bit più significativi; ma in un altro contesto, ***DS*:*SI*** può fare riferimento a un indirizzo che si interpreta come ***DS***·16+***SI***. Nello stesso modo, il numero rappresentato come 1000_{16}:$59DF_{16}$, potrebbe indicare precisamente il valore $100059DF_{16}$, oppure, se si tratta di un indirizzo, composto da segmento e scostamento, andrebbe inteso come $159DF_{16}$.

Registri

I registri dei microprocessori x86-16 sono schematizzati dalla figura successiva. I registri per uso generale, denominati ***AX***, ***BX***, ***CX*** e ***DX***, possono essere utilizzati nella loro interezza o divisi in byte; per esempio si può intervenire nel byte meno significativo di ***AX*** con il nome ***AL*** (*low*) e si può accedere al byte più significativo con il nome ***AH*** (*high*).

Figura u139.2. I registri dei microprocessori x86-16.



I registri di segmento sono: ***CS***, ***DS***, ***SS*** e ***ES***. Il segmento individuato dal registro ***CS*** (*code segment*) è quello in cui si svolge il codice in corso di esecuzione, e il puntatore all'istruzione da eseguire, nell'ambito del segmento codice, è contenuta nel registro ***IP*** (*instruction pointer*, ma noto anche come *program counter* e indicato a volte con la sigla «PC»). Il segmento individuato dal registro ***SS*** (***stack segment***) è quello in cui si trova la pila dei dati, ovvero quella struttura che consente il trasferimento delle variabili alle funzioni o la creazione di variabili locali. L'indice della pila è costituito dal registro ***SP*** (*stack pointer*) e l'indirizzo della base della pila, nell'ambito della funzione in corso di esecuzione, viene annotato convenzionalmente nel registro ***BP*** (*base pointer*). Il segmento individuato dal registro ***DS*** (***data segment***) è quello in cui si trovano i dati correnti, mentre il segmento del registro ***ES*** (***extra segment***) riguarda un'area dati alternativa, utile soprattutto quando si vogliono fare dei trasferimenti di dati tra segmenti differenti.

Convenzionalmente è stato adottato il registro ***BP*** per annotare il riferimento all'inizio della pila di una funzione, per poter accedere agli argomenti attuali o alle variabili locali con un riferimento relativo a tale puntatore. Tuttavia, va osservato che il segmento a cui si riferisce il registro ***BP*** è quello dei dati, ovvero ***DS***, per cui, quando si utilizza ***BP*** per accedere al contenuto della pila, è indispensabile che ***DS*** sia uguale a ***SS***.

Il registro ***FLAGS*** raccoglie gli indicatori disponibili, come descritto nella tabella successiva. In alcuni documenti, tale registro è chiamato *program status word* e abbreviato come «PSW».

Tabella u139.3. Gli indicatori principali contenuti nel registro ***FLAGS***.

Indicatore (*flag*)	Bit	Descrizione
C *carry*	0	È l'indicatore del riporto per le operazioni con valori senza segno. In particolare si attiva dopo una somma che genera un riporto e dopo una sottrazione che richiede il prestito di una cifra (in tal caso si chiama anche *borrow*).
1	1	Riservato.
P *parity*	2	Si attiva quando l'ultima operazione produce un risultato i cui otto bit meno significativi contengono una quantità pari di cifre a uno.
0	3	Riservato.
A *auxiliary carry*	4	È un tipo di riporto ausiliario.
0	5	Riservato.
Z *zero*	6	Viene impostato dopo un'operazione che dà come risultato il valore zero.
S *sign*	7	Riproduce il bit più significativo di un valore, dopo un'operazione. Se il valore è da intendersi con segno, l'indicatore serve a riprodurre il segno stesso.
T *trace*	8	Se è attivo, fa in modo che il microprocessore possa funzionare un passo alla volta.
I *interrupt*	9	Se è attivo, le interruzioni hardware sono abilitate, diversamente risultano bloccate.
D *direction*	10	Si usa per automatizzare le operazioni relative alle stringhe. Se è a zero, indica che la scansione della memoria deve procedere incrementando gli indici; se invece è pari a uno, la scansione deve proseguire decrementando gli indici.
O *overflow*	11	È l'indicatore di traboccamento per le operazioni che riguardano valori con segno.

Indicatore (*flag*)	Bit	Descrizione
0	12	Riservato.
0	13	Riservato.
0	14	Riservato.
0	15	Riservato.

I registri che sono definiti «per usi generali», hanno comunque un ruolo predominante. Tra questi si includono anche ***SI*** e ***DI***:

Registro	Definizione	Scopo prevalente
AX	accumulatore	Usato soprattutto nei calcoli e per l'input e output. Nelle convenzioni di chiamata comuni, si usa ***AX*** per restituire un valore attraverso una funzione.
BX	base	Viene usato particolarmente come indice da sommare ad altri, per individuare una posizione in memoria.
CX	contatore	Usato come contatore nei cicli.
DX	dati	Si affianca a ***AX***, soprattutto nelle divisioni e moltiplicazioni.
SI	*source index*	Usato prevalentemente come indice dell'origine, nell'ambito di un segmento dati (***DS*** o ***ES***).
DI	*destination index*	Usato prevalentemente come indice della destinazione, nell'ambito di un segmento dati (***DS*** o ***ES***).

Trasferimento di dati tra due segmenti differenti

Il trasferimento di dati tra segmenti di memoria differenti richiede l'uso di istruzioni apposite, con cui il registro ***DS*** individua il segmento di origine e ***ES*** quello di destinazione. Viene mostrato un esempio, con una porzione di codice, che ha lo scopo di copiare un intero segmento, dall'indirizzo efficace 10000_{16}, a $1FFFF_{16}$ incluso, a partire dall'indirizzo 30000_{16}, fino a $3FFFF_{16}$. La notazione è quella «Intel».

```
cld             ; Azzera l'indicatore di direzione.
mov   ax, 3000h ; Assegna a ES il segmento di destinazione,
mov   es, ax    ; attraverso AX.
mov   ax, 1000h ; Assegna a DS il segmento di origine,
mov   ds, ax    ; attraverso AX.
mov   cx, 8000h ; Imposta il contatore a 32768.
mov   si, 0h    ; Indice iniziale nel segmento di origine.
mov   di, 0h    ; Indice iniziale nel segmento di
                ; destinazione.
rep             ; Ripete l'istruzione successiva finché
                ; CX != 0; riducendo CX di una unità a ogni
                ; ciclo.
movsw           ; Copia 16 bit da DS:SI a ES:DI,
                ; incrementando di due unità sia SI, sia DI
                ; (in base all'indicatore di direzione).
```

Va osservato che ***CX*** riceve inizialmente un valore pari a metà della dimensione di un segmento, perché la copia avviene a coppie di byte, ovvero a interi di 16 bit. Si può notare anche che i registri di segmento coinvolti ricevono il valore attraverso la mediazione di ***AX***, perché non gli si può assegnare direttamente un valore immediato.

Riferimenti a indirizzi di memoria con i registri

Per indicare un indirizzo di memoria, generalmente si può utilizzare una costante numerica pura e semplice, ovvero un valore immediato, ma spesso è possibile combinare il valore di uno o più registri. Nella notazione Intel, per specificare che il risultato di un'espressione rappresenta un indirizzo di memoria, la si racchiude tra parentesi quadre. Per esempio, '`-2[DX+SI]`' fa riferimento all'indirizzo di memoria efficace che si ottiene come ***DS***·16+***DX***+***SI***−2 (***DS*** partecipa in quanto si fa riferimento a un segmento e può trattarsi solo di quello dei dati). Le combinazioni ammissibili sono rappresentate dal modello seguente, tenendo conto che qui le parentesi quadre indicano un blocco opzionale:

[costante]+[BX|BP]+[SI|DI]

Lo specchietto successivo riepiloga tutte le combinazioni ammissibili, dove la sigla ***imm*** rappresenta un valore immediato (una costante numerica letterale) che può essere sia positivo, sia negativo:

Notazione	Indirizzo efficace corrispondente	Notazione	Indirizzo efficace corrispondente
`[SI]`	***DS***·16+***SI***	***imm***`[SI]`	***DS***·16+***SI***+***imm***
`[DI]`	***DS***·16+***DI***	***imm***`[DI]`	***DS***·16+***DI***+***imm***
`[BP]`	***DS***·16+***BP***	***imm***`[BP]`	***DS***·16+***BP***+***imm***
`[BX]`	***DS***·16+***BX***	***imm***`[BX]`	***DS***·16+***BX***+***imm***
`[BX+SI]`	***DS***·16+***BX***+***SI***	***imm***`[BX+SI]`	***DS***·16+***BX***+***SI***+***imm***
`[BX+DI]`	***DS***·16+***BX***+***DI***	***imm***`[BX+DI]`	***DS***·16+***BX***+***DI***+***imm***
`[BP+SI]`	***DS***·16+***BP***+***SI***	***imm***`[BP+SI]`	***DS***·16+***BP***+***SI***+***imm***
`[BP+DI]`	***DS***·16+***BP***+***DI***	***imm***`[BP+DI]`	***DS***·16+***BP***+***DI***+***imm***

Si osservi che la costante letterale che precede il gruppo tra parentesi quadre può essere sostituita da un nome simbolico, con il quale si indica una variabile in memoria (preferibilmente un array). In tal modo, la notazione richiama quella degli array, come si fa con il linguaggio C. Per esempio, '`x[SI]`', individua così il byte ***SI***-esimo a partire dall'indirizzo a cui si riferisce ***x***.

Convenzioni di chiamata

Le convenzioni di chiamata adottate per i microprocessori x86-16 sono le stesse di quelle usate per x86-32:

- si inseriscono nella pila dei dati gli argomenti della chiamata, in ordine inverso, in modo che l'ultimo inserimento sia quello del primo parametro della funzione;

  ```
  push …
  ```

- si esegue la chiamata;

  ```
  call …
  ```

- all'interno della funzione si salva il valore di ***BP*** nella pila, si assegna a ***BP*** l'indice attuale della pila (in modo da poter usare ***BP*** come riferimento per raggiungere nella pila gli argomenti della chiamata e le variabili locali) e si allocano nella stessa le variabili locali (variabili automatiche);

  ```
  enter …
  ```

- si salvano nella pila i registri che la funzione va a modificare, quindi si procede con il lavoro della funzione;

  ```
  pusha
  ```

- il primo argomento della chiamata si raggiunge con '`+4[BP]`', il secondo con '`+6[BP]`',... la prima variabile locale si raggiunge con '`-2[BP]`', la seconda con '`-4[BP]`',...

Al termine della funzione si fa in modo di ripristinare la situazione precedente alla chiamata, restituendo eventualmente un valore attraverso il registro ***AX*** o eventualmente la coppia ***DX***:***AX***;

- vengono ripristinati i registri salvati all'inizio della funzione;

  ```
  popa
  ```

- se la funzione deve restituire un valore viene, questo viene assegnato a ***AX***, oppure ***DX***:***AX*** (se questo valore è da 32 bit);

  ```
  mov ax, -m[bp]
  mov dx, -n[bp]
  ```

- viene ridotta la pila riportandone l'indice al valore di ***BP*** e recuperando il valore precedente di ***BP***;

  ```
  leave
  ```

• si ritorna all'indirizzo successivo alla chiamata;

```
ret
```

• si espellono gli argomenti della chiamata.

```
pop …
```

Sintesi delle istruzioni x86-16

Nelle tabelle successive vengono annotate le istruzioni che possono essere utilizzate con i microprocessori x86-16, raggruppate secondo il contesto a cui appartengono. Sono però escluse le istruzioni '**AA***x*' e '**DA***x*', relative alla gestione dei numeri in formato BCD (*Binary coded decimal*).

L'ordine in cui sono specificati gli operandi è quello «Intel», ovvero appare prima la destinazione e poi l'origine. Le sigle usate per definire i tipi di operandi sono: ***reg*** per «registro»; ***mem*** per «memoria»; ***imm*** per «immediato» (costante numerica).

Quando appare la pseudocodifica che deve spiegare l'effetto di un'istruzione, i riferimenti agli indirizzi in memoria vengono fatti in modo inusuale. Per esempio, ***(DS·16+SI)*** indica un indirizzo in memoria, individuato dal registro ***SI*** che si riferisce al segmento annotato in ***DS***. In modo analogo, ****(DS·16+SI)*** individua il contenuto della memoria al tale indirizzo, mentre ***&nome*** rappresenta l'indirizzo in memoria del simbolo ***nome***.

Nella colonna degli indicatori appare: il simbolo «#» per annotare che l'indicatore relativo può essere modificato dall'istruzione; il simbolo «t» per annotare che lo stato precedente dell'indicatore viene considerato dall'istruzione; zero o uno se l'indicatore viene impostato in un certo modo; il simbolo «?» se l'effetto dell'istruzione sull'indicatore è indefinito.

Tabella u139.16. Assegnamenti, scambi, conversioni e istruzione nulla.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NOP`		*not operate* Istruzione nulla.	`cpazstido` `.........`
`MOV`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*move* Copia il valore dell'origine nella destinazione. Consente la copia da e verso i registri di segmento, ma per assegnare un valore a un registro di segmento occorre eseguire un passaggio intermedio attraverso un registro per usi generali. Origine e destinazione devono avere la stessa quantità di bit. ***dst*** `:=` ***org***	`cpazstido` `.........`
`LEA`	***reg***, ***mem***	*load effective address* Mette nel registro l'indirizzo della memoria, inteso come scostamento dall'inizio del segmento dati. ***dst*** `:=` ***&org***	`cpazstido` `.........`
`LDS`	***reg***, ***mem***	*load pointer using DS* Carica dalla memoria un valore a 32 bit, diviso in due blocchi da 16 bit, mettendo il primo blocco nel registro indicato e mettendo il secondo nel registro ***DS*** (segmento dati). ***DS*** `:` ***dst*** `:=` ***org***	`cpazstido` `.........`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`LES`	***reg***, ***mem***	*load pointer using ES* Carica dalla memoria un valore a 32 bit, diviso in due blocchi da 16 bit, mettendo il primo blocco nel registro indicato e mettendo il secondo nel registro ***ES***. ***ES*** `:` ***dst*** `:=` ***org***	`cpazstido` `.........`
`XCHG`	***reg***, ***reg*** ***reg***, ***mem*** ***mem***, ***reg***	*exchange data* Scambia i valori. ***dst*** `:==:` ***org***	`cpazstido` `.........`
`CBW`		*convert byte to word* Converte un intero con segno, della dimensione di 8 bit, contenuto in ***AL***, in modo da occupare tutto ***AX*** (da 8 bit a 16 bit). L'espansione tiene conto del segno. ***AX*** `:=` ***AL***	`cpazstido` `.........`

`CWD`		*convert word to double word*
		Converte un intero con segno, della dimensione di 16 bit, contenuto in ***AX***, in modo da estendersi anche in ***DX***, tenendo conto del segno.
		`IF` ***AX*** `>= 0`
		`THEN`
		`  ` ***DX*** `:= 0`
		`ELSE`
		`  ` ***DX*** `:=` $\texttt{FFFF}_{16}$
			`cpazstido`
			`.........`

`LAHF`		*load flags into AH* Carica i primi otto indicatori in ***AH***, escludendo quelli riservati. AH_{bit0}`:=`***c*** AH_{bit1}`:=1` AH_{bit2}`:=`***p*** AH_{bit3}`:=0` AH_{bit4}`:=`***a*** AH_{bit5}`:=0` AH_{bit6}`:=`***z*** AH_{bit7}`:=`***s***	`cpazstido` `#########`
`SAHF`		*store AH into flags* Modifica il valore dei primi otto indicatori, esclusi i bit 1, 3 e 5 (il secondo, il quarto e il sesto, che sono riservati e a loro non si attribuisce un significato particolare), scrivendoci sopra il contenuto di ***AH***. ***c*** `:=`AH_{bit0} ***p*** `:=`AH_{bit2} ***a*** `:=`AH_{bit4} ***z*** `:=`AH_{bit6} ***s*** `:=`AH_{bit7}	`cpazstido` `#########`

Tabella u139.17. Movimento di dati.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

LODSB		*load string byte*
		Dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), viene letto un byte e copiato in ***AL***. Se l'indicatore di direzione è pari a zero, ***SI*** viene incrementato di una unità, altrimenti viene decrementato di una unità.
		AL `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`++`
		`ELSE`
		`    `***SI***`--`
			`cpazstido`
			`.......t.`
			`.........`

LODSW		*load string word*
		Dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), viene letto un blocco da 16 bit e copiato in ***AX***. Se l'indicatore di direzione è pari a zero, ***SI*** viene incrementato di due unità, altrimenti viene decrementato di due unità.
		AL `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`:+=2`
		`ELSE`
		`    `***SI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

STOSB		*store string byte*
		All'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), viene scritto il valore contenuto in ***AL***, aggiornando ***DI*** in base al contenuto dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ***AL***
		`IF` ***d***`==0`
		`THEN`
		`    `***DI***`++`
		`ELSE`
		`    `***DI***`--`
			`cpazstido`
			`.......t.`
			`.........`

STOSW		*store string word*
		All'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), viene scritto il valore contenuto in ***AX***, aggiornando ***DI*** in base al contenuto dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ***AX***
		`IF` ***d***`==0`
		`THEN`
		`    `***DI***`:+=2`
		`ELSE`
		`    `***DI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

MOVSB		*move string byte*
		Copia un byte, dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), all'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando ***SI*** e ***DI*** in base al valore dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`++`
		`    `***DI***`++`
		`ELSE`
		`    `***SI***`--`
		`    `***DI***`--`
			`cpazstido`
			`.......t.`
			`.........`

MOVSW		*move string word*
		Copia un blocco di 16 bit, dall'indirizzo a cui punta la coppia ***DS:SI*** (***DS·16+SI***), all'indirizzo a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando ***SI*** e ***DI*** in base al valore dell'indicatore di direzione.
		****(ES·16+DI)*** `:=` ****(DS·16+SI)***
		`IF` ***d***`==0`
		`THEN`
		`    `***SI***`:+=2`
		`    `***DI***`:+=2`
		`ELSE`
		`    `***SI***`:-=2`
		`    `***DI***`:-=2`
			`cpazstido`
			`.......t.`
			`.........`

REP		*repeat*
		Ripete l'istruzione successiva (che può essere una tra: '**`LODSB`**', '**`LODSW`**', '**`STOSB`**', '**`STOSW`**', '**`MOVSB`**', '**`MOVSW`**'), per ***CX*** volte.
		`IF` ***CX***`!=0`
		`THEN`
		`    istruzione successiva`
		`    `***CX***`--`
		`ELSE`
		`    break`
			`cpazstido`
			`.........`

XLATB		*translate table to byte* Assegna a ***AL*** il valore che si può raggiungere all'indirizzo composto da ***DS***:***BX***+***AL*** (***DS·16+BX+AL***), dove ***AL*** va inteso come valore senza segno. ***AL*** `:=` ****(DS·16+BX+AL)***	`cpazstido` `.........`

Tabella u139.18. Confronti con la memoria.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`SCASB`		*compare string byte*
		Confronta il contenuto di ***AL*** con il valore a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando di conseguenza gli indicatori e anche il registro ***DI*** in base all'indicatore di direzione.
		****(ES·16+DI) - AL***
		`IF d==0`
		`THEN`
		`  DI++`
		`ELSE`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

`SCASW`		*compare string word*
		Confronta il contenuto di ***AX*** con il valore a cui punta la coppia ***ES:DI*** (***ES·16+DI***), aggiornando di conseguenza gli indicatori e anche il registro ***DI*** in base all'indicatore di direzione.
		****(ES·16+DI) - AX***
		`IF d==0`
		`THEN`
		`  DI:+=2`
		`ELSE`
		`  DI:-=2`
			`cpazstido`
			`.......t.`
			`#####···#`

`CMPSB`		*compare string byte in memory*
		Confronta il byte a cui punta la coppia ***ES:DI*** (***ES·16+DI***), con quello a cui punta la coppia ***DS:SI*** (***DS·16+SI***), aggiornando di conseguenza gli indicatori e anche i registri ***DI*** e ***SI*** in base all'indicatore di direzione.
		****(DS·16+SI) - *(ES·16+DI)***
		`IF d==0`
		`THEN`
		`  SI++`
		`  DI++`
		`ELSE`
		`  SI++`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`CMPSW`		*compare string word in memory*
		Confronta il blocco da 16 bit a cui punta la coppia ***ES:DI*** (***ES·16+DI***), con quello a cui punta la coppia ***DS:SI*** (***DS·16+SI***), aggiornando di conseguenza gli indicatori e anche i registri ***DI*** e ***SI*** in base all'indicatore di direzione.
		****(DS·16+SI) - *(ES·16+DI)***
		`IF d==0`
		`THEN`
		`  SI++`
		`  DI++`
		`ELSE`
		`  SI++`
		`  DI--`
			`cpazstido`
			`.......t.`
			`#####···#`

`REPE`
`REPZ`
		repeat while equal
		repeat while zero
		Ripete l'istruzione successiva (che può essere una tra: '**`SCASB`**', '**`SCASW`**', '**`CMPSB`**', '**`CMPSW`**'), fino a che l'indicatore ***z*** è pari a uno (rappresentante l'uguaglianza di una comparazione, ovvero che la sottrazione dà zero), fino a un massimo di ***CX*** volte.
		`IF CX!=0`
		`THEN`
		`  istruzione successiva`
		`  CX--`
		`  IF z==1`
		`  THEN`
		`  continue`
		`  ELSE`
		`  break`
		`ELSE`
		`  break`
			`cpazstido`
			`...#.....`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

`REPNE`
`REPNZ`
		repeat while not equal
		repeat while not zero
		Ripete l'istruzione successiva (che può essere una tra: '**`SCASB`**', '**`SCASW`**', '**`CMPSB`**', '**`CMPSW`**'), fino a che l'indicatore ***z*** è pari a zero (rappresentante la disuguaglianza della comparazione, ovvero che la sottrazione non dà zero), fino a un massimo di ***CX*** volte.
		`IF CX!=0`
		`THEN`
		`  istruzione successiva`
		`  CX--`
		`  IF z==0`
		`  THEN`
		`  continue`
		`  ELSE`
		`  break`
		`ELSE`
		`  break`
			`cpazstido`
			`...#.....`

Tabella u139.19. Operazioni aritmetiche.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NEG`	***reg*** ***mem***	*negation* Inverte il segno di un numero, attraverso il complemento a due. ***operand*** `:=` ***−operand***	`cpazstido` `##·##···#`
`ADD`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*addition* Somma di interi, con o senza segno, ignorando il riporto precedente. Se i valori si intendono con segno, è importante l'esito dell'indicatore di traboccamento (*overflow*), se invece i valori sono da intendersi senza segno, è importante l'esito dell'indicatore di riporto (*carry*). ***dst*** `:=` ***org*** `+` ***dst***	`cpazstido` `##·##···#`
`SUB`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*subtraction* Sottrazione di interi con o senza segno, ignorando il riporto precedente. ***dst*** `:=` ***org*** `−` ***dst***	`cpazstido` `##·##···#`

`ADC`	***reg***, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		addition with carry
		Somma di interi, con o senza segno, aggiungendo anche il riporto precedente (l'indicatore *carry*).
		dst `:=` ***org*** `+` ***dst*** `+` ***c***
			`cpazstido`
			`t·····`
			`##·##···#`

`SBB`	***reg***, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		subtraction with borrow
		Sottrazione di interi, con o senza segno, tenendo conto del «prestito» precedente (l'indicatore *carry*).
		dst `:=` ***org*** `+` ***dst*** `−` ***c***
			`cpazstido`
			`t·····`
			`##·##···#`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`INC`	***reg*** ***mem***	*increment* Incrementa di una unità un intero. ***operand***`++`	`cpazstido` `·#·##···#`
`DEC`	***reg*** ***mem***	*decrement* Decrementa di una unità un valore intero. ***operand***`−−`	`cpazstido` `·#·##···#`
`MUL`	***reg*** ***mem***	*multiply* Moltiplicazione intera senza segno. L'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti. ***AX*** `:=` ***AL*** `*` ***operand*** ***DX:AX*** `:=` ***AX*** `*` ***operand***	`cpazstido` `#?·??···#`
`DIV`	***reg*** ***mem***	*division* Divisione intera senza segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti. ***AL*** `:=` ***AX*** `/` ***operand*** ***AH*** `:=` ***AX%operand*** ***AX*** `:=` ***(DX:AX)*** `/` ***operand*** ***DX*** `:=` ***(DX:AX)%operand***	`cpazstido` `??·??···?`
`IMUL`	***reg*** ***mem***	*signed multiply* Moltiplicazione intera con segno. In questo caso l'operando è il moltiplicatore, mentre il moltiplicando è costituito da registri prestabiliti. ***AX*** `:=` ***AL*** `*` ***operand*** ***(DX:AX)*** `:=` ***AX*** `*` ***operand***	`cpazstido` `#?·??···#`
`IDIV`	***reg*** ***mem***	*signed division* Divisione intera con segno. L'operando è il divisore, mentre il dividendo è costituito da registri prestabiliti. ***AL*** `:=` ***AX*** `/` ***operand*** ***AH*** `:=` ***AX%operand*** ***AX*** `:=` ***(DX:AX)*** `/` ***operand*** ***DX*** `:=` ***(DX:AX)%operand***	`cpazstido` `??·??···?`

Tabella u139.20. Operazioni logiche.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`NOT`	***reg*** ***mem***	NOT di tutti i bit dell'operando. ***dst*** `:= NOT` ***dst***	`cpazstido` `·····`

`AND`
`OR`
`XOR`
	reg, ***reg***
	reg, ***mem***
	reg, ***imm***
	mem, ***reg***
	mem, ***imm***
		AND, OR, o XOR, tra tutti i bit dei due operandi.
		dst `:=` ***org*** `AND` ***dst***
		dst `:=` ***org*** `OR` ***dst***
		dst `:=` ***org*** `XOR` ***dst***
			`cpazstido`
			`0#·##···0`

Tabella u139.21. Scorrimenti e rotazioni.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
SHL SHR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*shift left* *shift right* Fa scorrere i bit, rispettivamente verso sinistra o verso destra (l'ultima cifra perduta finisce nell'indicatore del riporto). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····#
SAL SAR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*shift arithmetically left* *shift arithmetically right* Fa scorrere i bit, rispettivamente verso sinistra o verso destra (l'ultima cifra perduta finisce nell'indicatore del riporto), mantenendo il segno originale (logicamente '**SAL**' è identico a '**SHL**'). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····
RCL RCR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*rotate left with carry* *rotate right with carry* Ruota i bit, rispettivamente verso sinistra o verso destra, utilizzando anche l'indicatore di riporto (*carry*). Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido t····· #·····#
ROL ROR	***reg***, 1 ***mem***, 1 ***reg*** ***mem***	*rotate left* *rotate right* Ruota i bit, rispettivamente verso sinistra o verso destra. Se appare un solo operando, la rotazione viene eseguita ***CL*** volte. Se il valore immediato è maggiore di uno, è il compilatore che ripete l'istruzione per più volte.	cpazstido #·····#

Tabella u139.22. Chiamate e gestione della pila.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CALL	***reg*** ***mem*** ***imm***	Inserisce nella pila l'indirizzo dell'istruzione successiva e salta all'indirizzo indicato, che si riferisce allo scostamento a partire dall'inizio del segmento codice (***CS***). Pertanto, l'indirizzo a cui ci si riferisce è a 16 bit. push ***indirizzo_successivo*** ***IP*** := ***operand***	cpazstido ·····

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CALL FAR	***imm***:***imm***	*call procedure* Inserisce nella pila il valore di ***CS*** e poi l'indirizzo dell'istruzione successiva (***IP*** dell'istruzione successiva) e salta all'indirizzo indicato. L'indirizzo deve essere di quattro byte (32 bit), in quanto deve specificare anche il segmento codice da raggiungere. push ***CS*** push ***indirizzo_successivo*** ***CS***:***IP*** := ***operand***	cpazstido ·····
RET		*return from call* Estrae dalla pila l'indirizzo dell'istruzione da raggiungere (***IP***) e salta a quella (serve a concludere una chiamata eseguita con '**CALL**'). pop ***IP***	cpazstido ·····
RETF RET FAR		*return from far call* Estrae dalla pila il valore di ***CS*** e quindi l'indirizzo dell'istruzione da raggiungere (***IP***) e salta a quella (serve a concludere una chiamata eseguita con '**CALL FAR**'). pop ***IP*** pop ***CS***	cpazstido ·····
PUSH	***reg*** ***mem***	*push data onto stack* Inserisce nella pila il valore (della dimensione di un registro comune). ***SP***:-=2 *(***SS***·16+***SP***):=***operand***	cpazstido ·····
POP	***reg*** ***mem***	*pop data from stack* Estrae dalla pila l'ultimo valore inserito (della dimensione di un registro comune). ***operand***:=*(***SS***·16+***SP***) ***SP***:+=2	cpazstido ·····
PUSHF		*push flags onto stack* Inserisce nella pila l'insieme del registro degli indicatori (***FLAGS***). push ***FLAGS***	cpazstido ·····
POPF		*pop flags from stack* Estrae dalla pila l'insieme del registro degli indicatori (***FLAGS***), aggiornando di conseguenza il registro stesso. pop ***FLAGS***	cpazstido ?????????

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
ENTER	***imm8***, 0	*enter stack frame* Questa funzione esiste a partire dai microprocessori i186. Inserisce nella pila il valore di ***BP***, poi assegna a ***BP*** il valore di ***SP*** e infine decrementa ***SP*** del valore fornito come immediato. Serve a predisporre ***BP*** e ***SP*** all'inizio di una funzione, specificando lo spazio necessario per le variabili locali nella pila. `push` ***BP*** ***BP***`:=`***SP*** ***SP***`:-=2*`***dst***	`cpazstido` `.........`
PUSHA		*push all registers onto stack* Questa funzione esiste a partire dai microprocessori i186. Inserisce nella pila i registri principali: ***AX***, ***CX***, ***DX***, ***BX***, ***SP***, ***BP***, ***SI***, ***DI***. `push` ***AX*** `push` ***CX*** `push` ***DX*** `push` ***BX*** `push` ***SP*** `push` ***BP*** `push` ***SI*** `push` ***DI***	`cpazstido` `.........`
POPA		*pop all registers from stack* Questa funzione esiste a partire dai microprocessori i186. Ripristina i registri principali, estraendo i contenuti dalla pila: ***DI***, ***SI***, ***BP***, ***SP*** viene eliminato senza aggiornare il registro, ***BX***, ***DX***, ***CX***, ***AX***. Come si vede, anche se '**`PUSHA`**' salva l'indice della pila, in pratica questo indice non viene ripristinato. `pop` ***DI*** `pop` ***SI*** `pop` ***BP*** ***SP***`:+=2` `pop` ***BX*** `pop` ***DX*** `pop` ***CX*** `pop` ***AX***	`cpazstido` `.........`
LEAVE		*leave stack frame* Questa funzione esiste a partire dai microprocessori i186. Ripristina i valori di ***BP*** e di ***SP***, allo stato che avevano prima dell'uso dell'istruzione '**`ENTER`**'. ***SP***`:=`***BP*** `pop` ***BP***	`cpazstido` `.........`

Tabella u139.23. Interruzioni.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori

INT	***imm8***	*call to interrupt*
		Esegue una chiamata attraverso un'interruzione. Prima di saltare alla codice relativo all'interruzione selezionata, inserisce nella pila ***FLAGS***, ***CS*** e ***IP***. Azzera anche l'indicatore ***IF*** (*interrupt flag*), mentre gli altri indicatori rimangono inalterati.
		`pusf`
		`push` ***CS***
		`push` ***IP***
		i`:=0`
		`jmp far 0:(`***operand***`·4)`
			`cpazstido`
			`.........`

IRET		*return from interrupt* Conclude l'esecuzione del codice relativo a un'interruzione recuperando dalla pila i valori inseriti alla chiamata con '**`INT`**': ***IP***, ***CS*** e ***FLAGS***. Pertanto, il valore degli indicatori viene ripristinato allo stato precedente alla chiamata. `pop` ***IP*** `pop` ***CS*** `popf`	`cpazstido` `##·##···#`
CLI		*clear interrupt flag* Azzera l'indicatore di abilitazione delle interruzioni (*interrupt flag*), disabilitando di conseguenza le interruzioni hardware. ***i***`:=0`	`cpazstido` `......0..`
STI		*set interrupt flag* Attiva l'indicatore di abilitazione delle interruzioni (*interrupt flag*), abilitando di conseguenza le interruzioni hardware. ***i*** `:= 1`	`cpazstido` `......1..`
HLT		*enter halt state* Ferma il sistema, fino a quando viene ricevuta un'interruzione hardware.	`cpazstido` `.........`

INTO		*interrupt if overflow*
		Se l'indicatore di straripamento è attivo, esegue la chiamata dell'interruzione numero 4 (la quale dovrebbe gestore il problema).
			`cpazstido`
			`........t`
			`.........`

Tabella u139.24. Indicatori e confronti tra registri.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
CLC		*clear carry flag* Azzera l'indicatore del riporto (*carry*), senza intervenire negli altri indicatori. ***c***`:=0`	`cpazstido` `0.......`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`CLD`		*clear direction flag* Azzera l'indicatore di direzione (*direction*), senza intervenire negli altri indicatori. ***d***`:=0`	`cpazstido` `·····0·`
`STC`		*set carry flag* Attiva l'indicatore di riporto (*carry*). ***c***`:=1`	`cpazstido` `1·····`
`STD`		*set direction flag* Attiva l'indicatore di direzione (*direction*). ***d***`:=1`	`cpazstido` `·····1·`
`CMC`		*complement carry flag* Inverte il valore dell'indicatore del riporto (*carry*).	`cpazstido` `#·····`
`CMP`	***reg***, ***reg*** ***reg***, ***mem*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*compare operands* Confronta due valori interi. La comparazione avviene simulando la sottrazione dell'origine dalla destinazione, senza però modificare gli operandi, ma aggiornando gli indicatori, come se fosse avvenuta una sottrazione vera e propria. ***dst*** `-` ***org***	`cpazstido` `##·##···#`
`TEST`	***reg***, ***reg*** ***reg***, ***imm*** ***mem***, ***reg*** ***mem***, ***imm***	*logical compare* AND dei due valori senza conservare il risultato. Serve solo a ottenere l'aggiornamento degli indicatori. ***dst*** `AND` ***org***	`cpazstido` `0#·##···0`

Tabella u139.25. Salti.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`JMP`	***reg*** ***mem*** ***imm***	*jump* Salto incondizionato all'indirizzo indicato, che si intende relativo al segmento codice (***CS***). ***IP*****`:=`*****operand***	`cpazstido` `·····`
`JMP FAR`	***imm***`:`***imm***	*far jump* Salto incondizionato all'indirizzo indicato, costituito sia dal segmento codice, sia dall'indirizzo relativo, all'interno di questo. ***CS*****`:`*****IP*****`:=`*****operand***	`cpazstido` `·····`
`JA` `JNBE`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era maggiore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
`JAE` `JNB`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era maggiore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JB` `JNAE`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JBE` `JNA`	***imm***	*conditional jump* Dopo un confronto di valori senza segno, salta se la destinazione era minore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `t··t····`
`JE`	***imm***	*conditional jump* Dopo un confronto, indipendentemente dal segno, salta se la destinazione era uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `==` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···t····`
`JNE`	***imm***	*conditional jump* Dopo un confronto, indipendentemente dal segno, salta se la destinazione era diversa dall'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `!=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···t····`
`JG` `JNLE`	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era maggiore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
JGE JNL	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era maggiore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `>=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JL JNGE	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era minore dell'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JLE JNG	***imm***	*conditional jump* Dopo un confronto con segno, salta se la destinazione era minore o uguale all'origine. Il salto riguarda solo l'ambito del segmento codice attuale. `CMP` ***dst***`,` ***org*** `IF` ***dst*** `<=` ***org*** `THEN` `go to` ***imm***	`cpazstido` `···tt···t`
JC JNC	***imm***	*conditional jump* Salta se l'indicatore del riporto (*carry*), rispettivamente, è attivo, oppure non è attivo. Il salto riguarda solo l'ambito del segmento codice attuale.	`cpazstido` `t·····`
JO JNO	***imm***	*conditional jump* Salta se l'indicatore di traboccamento (*overflow*), rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `·····t`
JS JNS	***imm***	*conditional jump* Salta se l'indicatore di segno (*sign*), rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `····t···`
JZ JNZ	***imm***	*conditional jump* Salta se l'indicatore di zero, rispettivamente, è attivo, oppure non è attivo.	`cpazstido` `···t····`
JP JPE	***imm***	*conditional jump* Salta se l'indicatore di parità è attivo.	`cpazstido` `·t·····`
JNP JPO	***imm***	*conditional jump* Salta se l'indicatore di parità non è attivo.	`cpazstido` `·t·····`
JCXZ	***imm***	*conditional jump* Salta se il valore contenuto nel registro ***CX*** è pari a zero.	`cpazstido` `·····`

Tabella u139.26. Iterazioni

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
LOOP	***imm8***	*loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `·····`
LOOPE LOOPZ	***imm8***	*conditional loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero e l'indicatore «zero» è attivo, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `···t·····`
LOOPNE LOOPNZ	***imm8***	*conditional loop* Senza alterare gli indicatori, decrementa di una unità il registro '**CX**', quindi, se il registro è ancora diverso da zero e l'indicatore «zero» non è attivo, salta all'indirizzo cui fa riferimento l'operando. Tale indirizzo non può essere molto lontano dalla posizione corrente.	`cpazstido` `···t·····`

Tabella u139.27. Input e output.

Nome	Operandi: ***dst***, ***org1***, ***org2***	Descrizione	Indicatori
IN	***AL***, ***imm8*** ***AX***, ***imm8***	*input* Assegna a ***AL*** o ***AX*** il valore letto dalla porta specificata; in tal caso il numero di porta non può essere superiore a 255.	`cpazstido` `·····`
IN	***AL***, ***DX*** ***AX***, ***DX***	*input* Assegna a ***AL*** o ***AX*** il valore letto dalla porta specificata da ***DX***; in tal caso il numero di porta può essere superiore a 255.	`cpazstido` `·····`
OUT	***imm8***, ***AL*** ***imm8***, ***AX***	*output* Scrive nella porta specificata il valore contenuto in ***AL*** o ***AX***; in tal caso il numero di porta non può essere superiore a 255.	`cpazstido` `·····`
OUT	***DX***, ***AL*** ***DX***, ***AX***	*output* Scrive nella porta indicata da ***DX*** il valore contenuto in ***AL*** o ***AX***; in tal caso il numero di porta può essere superiore a 255.	`cpazstido` `·····`

Sostituzione delle istruzioni per i186

Nella sezione precedente sono state menzionate delle istruzioni che non fanno parte dei microprocessori 8086/8088, ma queste possono essere ottenute facilmente attraverso altre istruzioni elementari,

tanto che l'assemblatore potrebbe provvedervi direttamente. A ogni modo viene annotato qui come possono essere sostituite.

Listato u139.28. Sostituzione per l'istruzione '**PUSHA**'.

```
push ax
push cx
push dx
push bx
push sp
push bp
push si
push di
```

Listato u139.29. Sostituzione per l'istruzione '**POPA**'. Il registro ***SP*** non viene ripristinato, di conseguenza si riduce l'indice della pila (si incrementa ***SP***) senza estrarne il valore.

```
pop di
pop si
pop bp
add sp, 2       ; non ripristina SP
pop bx
pop dx
pop cx
pop ax
```

Listato u139.30. Sostituzione per l'istruzione '**ENTER**'. La riduzione di ***SP*** dipende dalla quantità di variabili locali che si vogliono gestire. Usando interi da 16 bit, si tratta di moltiplicare la quantità di variabili locali per due. Va ricordato che il segmento a cui si riferisce ***BP*** è ***DS***, per cui è indispensabile che ***DS*** sia uguale a ***SS***, essendo usato in questo modo come riferimento alla pila.

```
push bp
mov  bp, sp
sub  sp, 2      ; 0, 2, 4, 6,...
```

Listato u139.31. Sostituzione per l'istruzione '**LEAVE**'.

```
mov sp, bp
pop bp
```

Riferimenti

- Andrew S. Tanenbaum, *Operating Systems: Design and Implementation*, **prima edizione**, 1987, Prentice-Hall, ISBN 0-13-637406-9

  Appendice B: *introduction to the IBM PC*
- MAD, *Assembly tutorial*
  http://www.xs4all.nl/~smit/asm01001.htm
- Wikipedia, *x86 instruction listings*
  http://en.wikipedia.org/wiki/X86_instruction_listings
- *The x86 Interrupt List, aka "Ralf Brown's Interrupt List", "RBIL"*
  http://www.cs.cmu.edu/~ralf/files.html
- *Computer interrupt*
  http://wayback.archive.org/web/20040101000000/http://calab.kaist.ac.kr/~hyoon/courses/cs310_2001fa01ll/micro17.ppt*
  http://www.ece.msstate.edu/~reese/EE3724/lectures/interrupt/interrupt.pdf
- BiosCentral, *BIOS data area*
  http://www.bioscentral.com/misc/bda.htm
- Robert de Bath, *Linux 8086 development environment*
  http://homepage.ntlworld.com/robert.debath/
  http://homepage.ntlworld.com/robert.debath/dev86/

Architettura IBM PC

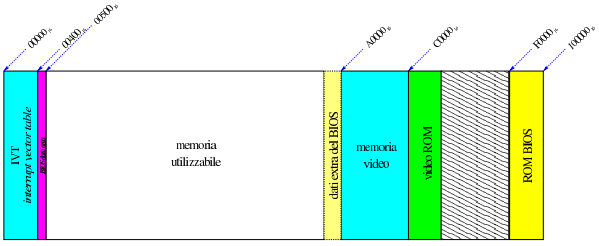
IVT: «interrupt vector table» 1307
BIOS data area 1308
Altre aree di memoria 1309
Interruzioni principali 1309
- INT 10_{16}/AH=00_{16} «set video mode» 1309
- INT 10_{16}/AH=05_{16} «select active display page» 1309
- INT 10_{16}/AH=$0E_{16}$ «teletype output» 1309
- INT 12_{16} «get memory size» 1310
- INT 13_{16}/AH=00_{16} «reset disk system» 1310
- INT 13_{16}/AH=02_{16} «read disk sectors into memory» 1310
- INT 13_{16}/AH=03_{16} «write disk sectors» 1310
- INT 16_{16}/AH=00_{16} «get keystroke from keyboard» 1311
- INT 16_{16}/AH=01_{16} «check for keystroke in the keyboard buffer» 1311

Riferimenti 1311

L'architettura del vecchio IBM PC prevede 1 Mibyte di memoria, in cui alcune fasce hanno degli scopi particolari e non possono essere utilizzate diversamente. Quando si programma a 32 bit, di norma si dispone, in proporzione, di una quantità enorme di memoria, per cui di solito lo spazio inferiore a 1 Mibyte viene semplicemente ignorato e si considera solo lo spazio successivo; ma se la programmazione avviene a 16 bit, si deve operare in quello spazio ristretto.

Buona parte della memoria di un elaboratore conforme all'architettura del IBM PC è impegnata per il codice del BIOS, in cui sono contenute diverse routine attivate da interruzioni, hardware o software. Nella limitazione dell'architettura originale, il sistema offerto di gestione delle interruzioni consentirebbe un utilizzo uniforme dell'hardware.

Figura u140.1. Mappa generale della memoria.

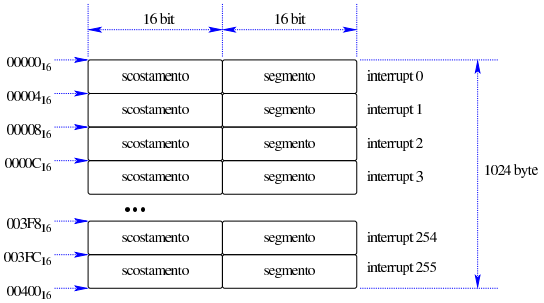


IVT: «interrupt vector table»

I microprocessori x86-16 utilizzano una tabella delle interruzioni collocata nella prima parte della memoria, a partire dall'indirizzo 0. La tabella è nota con la sigla IVT, ovvero *Interrupt vector table*.

Le voci della tabella IVT sono costituite semplicemente da due informazioni: l'indirizzo di segmento e lo scostamento in cui l'interruzione rispettiva viene gestita, attraverso del codice appropriato. Va osservato però che appare prima lo scostamento e poi il segmento.

Figura u140.2. Tabella IVT in memoria.



Supponendo sia ***n*** il numero di un'interruzione, si ottiene l'indirizzo della routine di interruzione con un calcolo molto semplice: lo scostamento è disponibile all'indirizzo ***n***×4; il segmento è disponibile all'indirizzo ***n***×4+2.

Solo una porzione delle voci della tabella è utilizzata in modo predefinito, mentre le altre sono disponibili per interruzioni aggiuntive a uso del sistema operativo. In ogni caso, tutta la tabella può essere modificata, in modo da dirigere le interruzioni a codice alternativo.

Tabella u140.3. Configurazione iniziale della tabella IVT, secondo l'architettura IBM PC.

Interruzione	Descrizione
00_{16}	Eccezione del microprocessore: *divide error*.
01_{16}	Eccezione del microprocessore: *debug*.
02_{16}	Eccezione del microprocessore: *non maskable interrupt*.
03_{16}	Eccezione del microprocessore: *breakpoint*.
04_{16}	Eccezione del microprocessore: *arithmetic overflow*.
05_{16}	BIOS: il tasto [*Stampa*] o [*Print_Screen*] è stato premuto.
06_{16}	Eccezione del microprocessore: *invalid opcode*.
07_{16}	Eccezione del microprocessore: *no coprocessor*.
08_{16}	Interruzione hardware: IRQ 0, temporizzatore.
09_{16}	Interruzione hardware: IRQ 1, tastiera.
$0A_{16}$	Interruzione hardware: IRQ 2.
$0B_{16}$	Interruzione hardware: IRQ 3.
$0C_{16}$	Interruzione hardware: IRQ 4.
$0D_{16}$	Interruzione hardware: IRQ 5.
$0E_{16}$	Interruzione hardware: IRQ 6.
$0F_{16}$	Interruzione hardware: IRQ 7.
10_{16}	BIOS: video.
11_{16}	BIOS: controllo dell'apparecchiatura.
12_{16}	BIOS: memoria disponibile.
13_{16}	BIOS: unità a disco.
14_{16}	BIOS: porte seriali.
15_{16}	BIOS: funzioni varie.
16_{16}	BIOS: tastiera.
17_{16}	BIOS: porte parallele.
18_{16}	BIOS: interprete BASIC in ROM.
19_{16}	BIOS: riavvio.
$1A_{16}$	BIOS: orologio.
$1B_{16}$	BIOS: tasto [*Interr*] o [*Break*].
$1C_{16}$	BIOS: temporizzatore.
$1D_{16}$	BIOS: inizializzazione video.
$1E_{16}$	BIOS: inizializzazione gestione dischi.
$1F_{16}$	BIOS: caratteri del video.

Quando si costruisce una procedura da associare a una voce della tabella delle interruzioni, occorre considerare che prima che la procedura stessa sia raggiunta, il microprocessore inserisce nella pila delle informazioni. Nello specchietto successivo si mettono a confronto le istruzioni relative alle chiamate di interruzioni e di procedure comuni:

`int`	`call far`	`call`

`pushf`
`push cs`
`push ip`
	`push cs`
	`push ip`
		`push ip`

`iret`	`retf`	`ret`

`pop ip`
`pop cs`
`popf`
	`pop ip`
	`pop cs`
		`pop ip`

BIOS data area

Dopo la tabella IVT, a partire dall'indirizzo 00400_{16} e fino a $004FF_{16}$ incluso, si trova un'area di memoria utilizzata dal BIOS, per annotarvi delle informazioni. Alla fine del capitolo sono riportati i riferimenti alla documentazione che consente di interpretare il contenuto di questa area, ma quello che conta sapere è che non ci si deve scrivere, a meno di impedire alle funzioni del BIOS di operare correttamente.

Altre aree di memoria

A partire dall'indirizzo $A0000_{16}$, fino a $BFFFF_{16}$ incluso, si trova la memoria usata per rappresentare i dati sullo schermo. Successivamente ci sono altre aree di memoria in sola lettura (un'area precedente all'indirizzo $F0000_{16}$ potrebbe essere priva di qualunque cosa) in particolare il blocco da $F0000_{16}$ a $FFFFF_{16}$ che contiene le procedure del BIOS.

In pratica, a parte la possibilità di scrivere direttamente nella memoria video, per ottenere la rappresentazione del testo sullo schermo, la memoria da $A0000_{16}$ fino alla fine, non può essere utilizzata, ma rimane incerta una porzione di memoria antecedente l'indirizzo $A0000_{16}$ che potrebbe essere utilizzata anch'essa dalle procedure che compongono il BIOS.

Per conoscere l'ammontare di memoria libera si può leggere il valore contenuto all'indirizzo 00413_{16}, nell'ambito della *BIOS data area*, tenendo conto che si tratta di un numero a 16 bit. Quel valore indica la quantità di memoria utile, espressa in multipli di 1024 byte, ma occorre considerare che si può utilizzare solo a partire dall'indirizzo 00500_{16}, ovvero dalla fine della BDA. In alternativa, si può chiamare l'interruzione 12_{16}, ottenendo dal registro ***AX*** tale valore.

Interruzioni principali

Il BIOS di un elaboratore IBM PC offre una serie di funzionalità, attraverso delle interruzioni, le quali possono essere utilizzate in mancanza di funzioni più sofisticate del sistema operativo. Il testo di riferimento per le interruzioni, del BIOS e dei sistemi operativi che le estendono, è quello di Ralf Brown, annotato alla fine del capitolo. Nelle sezioni successive vengono descritte solo alcune interruzioni offerte esclusivamente da BIOS standard.

INT 10_{16}/AH=00_{16} «set video mode»

Definisce la modalità di funzionamento del video. A seconda della modalità scelta, si possono usare una o più «pagine» distinte.

Parametro	Descrizione
AH	00_{16}
AL	Modalità video: 00_{16} testo, 40×25 caratteri, 16 colori, 8 pagine; 03_{16} testo, 80×25 caratteri, 16 colori, 4 pagine.

INT 10_{16}/AH=05_{16} «select active display page»

Seleziona la pagina video attiva. La numerazione parte da zero e la quantità di pagine disponibili dipende dalla modalità scelta.

Parametro	Descrizione
AH	05_{16}
BH	Pagina video da selezionare.

INT 10_{16}/AH=$0E_{16}$ «teletype output»

Mostra un carattere sullo schermo, alla pagina specificata, facendo avanzare il cursore e facendo scorrere il testo precedente se necessario. In questa modalità di visualizzazione, i caratteri con funzioni speciali vengono interpretati secondo la tradizione, tenendo conto che <*CR*> riporta il cursore all'inizio della stessa riga e che <*LF*> fa avanzare alla riga successiva, ma senza riportare il cursore all'inizio.

Parametro	Descrizione
AH	$0E_{16}$
AL	Carattere da rappresentare.
BL	Numero della pagina video.

INT 12_{16} «get memory size»

Restituisce la dimensione della memoria utilizzabile, partendo dall'indirizzo 00000_{16}, espressa in multipli di 1024 byte. La memoria utilizzabile effettivamente inizia solo a partire dall'indirizzo 00500_{16}. Eventualmente lo stesso valore sarebbe accessibile all'indirizzo 00413_{16}, leggendo un numero da 16 bit.

Valore restituito	Descrizione
AX	Dimensione della memoria disponibile.

INT 13_{16}/AH=00_{16} «reset disk system»

Azzera il sistema di gestione dei dischi, per l'unità indicata. L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80_{16} in su.

Parametro	Descrizione
AH	00_{16}
DL	Numero dell'unità da azzerare.

Valore restituito	Descrizione
AH	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.
c (*carry*)	Zero se ***AH*** è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.

INT 13_{16}/AH=02_{16} «read disk sectors into memory»

Legge uno o più settori dal disco alla memoria. Il numero del cilindro in cui si trova il settore iniziale viene indicato utilizzando il registro ***CH*** per gli otto bit meno significativi, mentre si aggiungono altri due bit, più significativi, dal registro ***CL***.

L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80_{16} in su.

Parametro	Descrizione
AH	02_{16}
DL	Numero dell'unità da cui leggere.
AL	Quantità di settori da leggere (deve essere maggiore di zero).
CH, CL_{bit6}, CL_{bit7}	Numero del cilindro del primo settore da leggere, costituito dai bit 7 e 6 del registro ***DL*** e dal registro ***CL*** (da ***0*** a 1023).
CL_{bit0}..CL_{bit5}	Numero del settore, relativo alla traccia, intesa come combinazione di cilindro e testina (da 1 a 63).
DH	Numero della testina (da 0 a 63, perché i due bit più significativi potrebbero essere attribuiti ad altre funzioni).
ES:***BX*** (***ES*16+BX***)	Puntatore all'area di memoria che deve ricevere i settori letti.

Valore restituito	Descrizione
AH	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.
c (*carry*)	Zero se ***AH*** è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.
AL	In presenza di un errore, riporta la quantità di settori letti correttamente.

INT 13_{16}/AH=03_{16} «write disk sectors»

Scrive uno o più settori dalla memoria nel disco. Il numero del cilindro in cui si trova il settore iniziale viene indicato utilizzando il registro ***CH*** per gli otto bit meno significativi, mentre si aggiungono altri due bit, più significativi, dal registro ***CL***.

L'unità è un numero da zero in su per i dischetti, mentre per i dischi fissi si parte da 80_{16} in su.

Parametro	Descrizione
AH	03_{16}
DL	Numero dell'unità in cui scrivere.
AL	Quantità di settori da scrivere (deve essere maggiore di zero).

Parametro	Descrizione
CH, CL_{bit6}, CL_{bit7}	Numero del cilindro del primo settore da scrivere, costituito dai bit 7 e 6 del registro ***DL*** e dal registro ***CL*** (da ***0*** a 1023).
CL_{bit0}..CL_{bit5}	Numero del settore, relativo alla traccia, intesa come combinazione di cilindro e testina (da 1 a 63).
DH	Numero della testina (da 0 a 63, perché i due bit più significativi potrebbero essere attribuiti ad altre funzioni).
ES:***BX*** (***ES*16+BX***)	Puntatore all'area di memoria da cui trarre i dati per la scrittura dei settori.

Valore restituito	Descrizione
AH	Stato: zero indica un risultato soddisfacente, altrimenti si tratta di un errore.
c (*carry*)	Zero se ***AH*** è pari a zero; altrimenti, in presenza di un errore, l'indicatore viene attivato.
AL	In presenza di un errore, riporta la quantità di settori scritti correttamente.

INT 16_{16}/AH=00_{16} «get keystroke from keyboard»

Legge un valore inserito dalla tastiera, eliminandolo dalla memoria tampone associata.

Parametro	Descrizione
AH	00_{16}

Valore restituito	Descrizione
AH	Il valore letto, secondo il codice usato dal BIOS.
AL	Il valore letto, tradotto in un carattere ASCII.

INT 16_{16}/AH=01_{16} «check for keystroke in the keyboard buffer»

Legge un valore inserito dalla tastiera, ma senza eliminarlo dalla memoria tampone associata.

Parametro	Descrizione
AH	01_{16}

Valore restituito	Descrizione
z (*zero*)	Zero se la lettura è avvenuta con successo; altrimenti, se la memoria tampone è vuota, l'indicatore risulta attivato.
AH	Il valore letto, secondo il codice usato dal BIOS.
AL	Il valore letto, tradotto in un carattere ASCII.

Riferimenti

- Andrew S. Tanenbaum, *Operating Systems: Design and Implementation*, **prima edizione**, 1987, Prentice-Hall, ISBN 0-13-637406-9

  Appendice B: *introduction to the IBM PC*
- MAD, *Assembly tutorial*
  http://www.xs4all.nl/~smit/asm01001.htm
- Wikipedia, *x86 instruction listings*
  http://en.wikipedia.org/wiki/X86_instruction_listings
- *The x86 Interrupt List, aka "Ralf Brown's Interrupt List", "RBIL"*
  http://www.cs.cmu.edu/~ralf/files.html
- *Computer interrupt*
  http://wayback.archive.org/web/20040101000000/http://calab.kaist.ac.kr/~hyoon/courses/cs310_2001fa01ll/micro17.ppt*
  http://www.ece.msstate.edu/~reese/EE3724/lectures/interrupt/interrupt.pdf
- BiosCentral, *BIOS data area*
  http://www.bioscentral.com/misc/bda.htm
- Robert de Bath, *Linux 8086 development environment*

http://homepage.ntlworld.com/robert.debath/
http://homepage.ntlworld.com/robert.debath/dev86/

Preparazione 1313
Bcc 1314
As86 1315
Ld86 1315
Bootblocks 1316
Riferimenti 1317

Per chi si avvale di un sistema operativo GNU, gli strumenti per sviluppare codice per x86-16 sono costituiti dalla raccolta nota con il nome Dev86, la quale mette assieme il compilatore C Bcc,[1] l'assemblatore As86 e il «collegatore» Ld86, oltre a una libreria C adatta per produrre applicazioni per ELKS (*Embeddable Linux kernel subset*).

Considerato che strumenti del genere sono utili, presumibilmente, per realizzare un programma autonomo (*stand alone*) o il kernel di un sistema operativo, un programma di avvio facilita molto il lavoro e consente di concentrare l'attenzione su ciò che si vuole realizzare veramente. Per questo motivo, nel capitolo viene anche preso in considerazione Bootblocks per l'avvio di un sistema operativo da dischetti con file system Minix 1.

Preparazione

In una distribuzione GNU/Linux Debian sono disponibili i pacchetti '**`bcc`**', '**`bin86`**' e '**`elks-libc`**' che forniscono il necessario per la compilazione, ma in un altro sistema GNU può essere necessario procurarsi il pacchetto sorgente Dev86, dal quale si ottiene ciò che serve.

Se si è costretti a partire dai sorgenti di Dev86, una volta scaricato il pacchetto, questo può essere espanso in una directory qualunque nell'elaboratore GNU, come mostrato dall'esempio seguente, dove però, successivamente, si possano acquisire i privilegi dell'utente '**`root`**':

```
$ tar xzvf Dev86src-0.16.17.tar.gz [Invio]
```

Si ottiene la directory '`dev86-0.16.17/`' che si articola ulteriormente. Terminata l'installazione occorre compilare questi sorgenti e installarli. In questo caso si prevede di installare Dev86 a partire da '`/opt/dev86/`':

```
$ cd dev86-0.16.17 [Invio]

$ make PREFIX=/opt/dev86/ [Invio]
```

Viene richiesto di intervenire su alcuni indicatori (*flag*); in generale dovrebbe andare bene ciò che viene proposto in modo predefinito:

```
  1) (ON)    Library of bcc helper functions
  2) (ON)    Minimal syscalls for BIOS level
  3) (ON)    Unix error functions
  4) (ON)    Management for /etc/passwd /etc/group /etc/utmp
  5) (OFF)   Linux-i386 system call routines GCC
  6) (ON)    GNU termcap routines
  7) (ON)    Bcc 386 floating point
  8) (ON)    Linux-i386 system call routines
  9) (ON)    Example kernel include files and syscall.dat
 10) (ON)    Malloc routines
 11) (ON)    Various unix lib functions
 12) (ON)    Msdos system calls
 13) (ON)    Regular expression lib
 14) (ON)    Stdio package
 15) (ON)    String and memory manipulation
 16) (ON)    Linux-8086 system call routines
 17) (ON)    Termios functions
 18) (ON)    Unix time manipulation functions.

Select config option to flip [or quit] > quit [Invio]
```

Al termine della compilazione si passa all'installazione, cominciando dalla creazione della directory '`/opt/dev86/`'. Per fare questo

occorrono i privilegi dell'utente '**root**':

```
$ su [Invio]

...

# mkdir -p /opt/dev86 [Invio]

# make install [Invio]
```

Bcc

Bcc[2] è un compilatore C tradizionale, ovvero fatto per la vecchia sintassi, nota con la sigla K&R. Tuttavia, con l'ausilio di un programma esterno (di norma si tratta di Unproto di Wietse Venema, incluso nella distribuzione Dev86), può compilare sorgenti scritti nella forma di un C standard, pur non potendo disporre di tutte le funzionalità di un compilatore aggiornato.

```
bcc [opzioni] file_c...
```

Tabella u141.2. Alcune opzioni per l'uso di Bcc.

Opzione	Descrizione
`-ansi`	Si avvale di un programma esterno per poter accettare un sorgente scritto secondo le convenzioni attuali del linguaggio C, pur nei limiti di quanto Bcc può poi elaborare.
`-0`	Produce un codice adatto per CPU 8086/8088.
`-S`	Produce un file in linguaggio assemblatore, da usare poi con l'assemblatore As86. In mancanza dell'opzione '**-S**' o '**-c**', si ottiene direttamente un file eseguibile, con l'intervento automatico di As86 e di Ld86.
`-c`	Produce un file oggetto, da utilizzare poi con il collegatore Ld86. In mancanza dell'opzione '**-S**' o '**-c**', si ottiene direttamente un file eseguibile, con l'intervento automatico di As86 e di Ld86.
`-o` ***nome***	Produce un file con il nome specificato.
`-I`	Non utilizza i percorsi predefiniti per l'inclusione dei file di intestazione.
`-I`***percorso***	L'opzione '**-I**', a cui si attacca un percorso, aggiunge quel percorso a quelli usati per l'inclusione dei file di intestazione. Possono essere specificati più percorsi ripetendo l'uso dell'opzione.

L'esempio seguente mostra la compilazione del file '`mio.c`', per produrre il file '`mio.s`', contenente il codice in linguaggio assemblatore. Per la compilazione, i file di intestazione vengono cercati esclusivamente in percorsi stabiliti: '`../include`' e '`../../include`'.

```
$ bcc -ansi -0 -S -o mio.s ↩
↪     -I -I../include -I../../include mio.c [Invio]
```

L'esempio successivo è simile, ma si produce il file oggetto '`mio.o`':

```
$ bcc -ansi -0 -c -o mio.s ↩
↪     -I -I../include -I../../include mio.c [Invio]
```

I nomi delle variabili e delle funzioni scritte in linguaggio C, si traducono nel linguaggio assemblatore in nomi preceduti dal trattino basso. Per esempio, la funzione ***main()***, diventa il simbolo '**_main**'. Per questa ragione, quando si scrivono porzioni di codice in linguaggio assemblatore da esportare, occorre ricordare di aggiungere un trattino basso all'inizio.

A meno di voler produrre programmi per il sistema operativo ELKS, il compilatore Bcc va utilizzato con l'opzione '**-c**', oppure '**-S**', per poter controllare i passaggi successivi, in particolare la fase di collegamento dei vari componenti.

As86

As86 è un assemblatore in linguaggio x86, adatto alla compilazione di quanto prodotto da Bcc. La sintassi usata da As86 è fondamentalmente quella Intel.

```
as86 [opzioni] file_s
```

Tabella u141.3. Alcune opzioni per l'uso di As86.

Opzione	Descrizione
`-0`	Compila nella modalità a 16 bit e avvisa quando incontra istruzioni che non sono adatte a CPU 8086/8088.
`-o` ***nome***	Produce un file oggetto con il nome specificato.
`-s` ***nome***	Produce un file di testo contenente l'elenco dei simboli individuati. Questa opzione può essere usata assieme a '**-o**'.
`-u`	Fa in modo che i simboli privi di una dichiarazione siano importati dall'esterno senza specificare il segmento.

L'esempio seguente rappresenta una situazione di utilizzo comune, in cui si produce il file oggetto '`mio.o`', a partire dal sorgente '`mio.s`':

```
$ as86 -u -0 -o mio.o mio.s [Invio]
```

Come precisato a proposito di Bcc, se si devono importare dei simboli dal codice C, occorre aggiungere un trattino basso all'inizio dei nomi.

Ld86

Ld86 è il «collegatore» (*linker*) associato a As86. La caratteristica di Ld86 è quella di poter produrre un eseguibile «impuro» (come viene definito nella sua pagina di manuale), per il quale il segmento usato dal codice è lo stesso usato per i dati (***CS***==***DS***==***SS***), oppure può tenere separati il codice e i dati in segmenti distinti (ma in tal caso vale ancora l'uguaglianza ***DS***==***SS***).

Nella pagina di manuale di Ld86 si parla di «I&D», ovvero di istruzioni e dati, che possono essere separati o meno.

```
ld86 [opzioni] file_o...
```

Tabella u141.4. Alcune opzioni per l'uso di Ld86.

Opzione	Descrizione
`-d`	Elimina l'intestazione dal file che va a essere generato. L'utilizzo di questa opzione implica l'uso di '**-s**'.
`-s`	Elimina i simboli.
`-o` ***nome***	Produce un file eseguibile con il nome specificato.
`-i`	Tiene separati il segmento usato dal codice rispetto a quello dei dati. In mancanza di questa opzione, il segmento è lo stesso.

L'esempio seguente mostra la creazione di un programma, privo di intestazione e di simboli, dove tutto viene così definito attraverso il codice in modo esplicito. In particolare, si presume che il file '`crt0.o`' sia realizzato in modo da definire esattamente la forma della prima parte del file eseguibile.

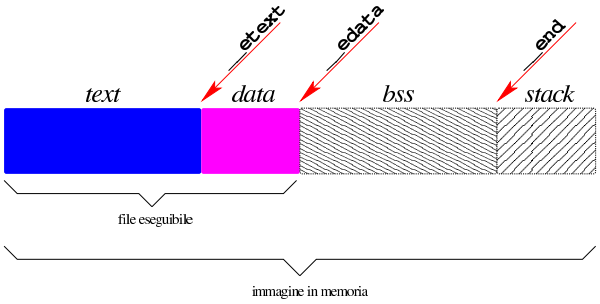
```
$ ld86 -d -s -o kimage crt0.o body.o main.o [Invio]
```

Ld86 definisce implicitamente dei simboli, raggiungibili dal codice che si scrive. Tra questi, sono molto importanti quelli seguenti, con cui è possibile determinare la collocazione in memoria del programma, distinguendo tra codice e dati:

Simbolo per As86	Nome per Bcc	Descrizione
`__etext`	`_etext`	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dal codice, nell'ambito del segmento in cui si colloca.
`__edata`	`_edata`	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dai dati inizializzati, nell'ambito del segmento in cui si colloca.
`__end`	`_end`	Variabile a 16 bit contenente l'indirizzo conclusivo dell'area usata dai dati non inizializzati (BSS), esclusa la pila, nell'ambito del segmento in cui si colloca.
`__segoff`	`_segoff`	Variabile a 16 bit contenente la distanza tra l'inizio del segmento codice e l'inizio di quello usato per i dati, espressa in multipli di 16 bit. In presenza di eseguibili in cui il segmento è lo stesso, questo valore è zero.

In generale, la struttura di un file eseguibile prodotto da Ld86 è composta inizialmente dal codice, quindi continua con i dati inizializzati. Lo spazio dei dati non inizializzati non fa parte del file e deve essere previsto quando si carica il file in memoria, per metterlo in esecuzione; inoltre, lo stesso va fatto per la pila (*stack*) dei dati, la quale deve collocarsi dopo tale area.

Figura u141.6. Contenuto di un programma privo di intestazione e di simboli.



Bootblocks

Il pacchetto Bootblocks[3] consente di avviare un sistema per elaboratori x86-16, contenuto in un dischetto o in una partizione del disco fisso, con il kernel inserito nello stesso file system. Il pacchetto viene distribuito assieme agli strumenti di sviluppo Dev86, ma non viene compilato automaticamente assieme a quelli. Si trova precisamente nella sottodirectory '`bootblocks/`' dei sorgenti di Dev86. Si compila in modo molto semplice con il comando '**`make`**':

```
# cd sorgenti_dev86/bootblocks [Invio]

# make [Invio]
```

Dalla compilazione si ottengono diversi file e sono utili in particolare:

File	Descrizione
'**`makeboot`**'	programma per l'installazione del settore di avvio, da usare attraverso un sistema GNU/Linux comune;
'**`makeboot.com`**'	programma analogo a '**`makeboot`**', da usare con un sistema Dos.

Per avviare un programma autonomo o un kernel vero e proprio, in un dischetto con file system Minix 1, è sufficiente copiare tale file in modo che si trovi nella directory principale con il nome '`boot`', oppure si crea la directory '`/boot/`' e vi si colloca il file con il nome che si preferisce.

Supponendo di utilizzare un sistema GNU/Linux, supponendo di

avere preparato il dischetto Minix (con i nomi al massimo di 14 byte) contenente tutto quello che serve, soprattutto con la directory '`/boot/`' o con il file '`boot`', se questo dischetto risulta inserito nell'unità corrispondente al file di dispositivo '`/dev/fd0`', **senza essere stato innestato**, si può eseguire il comando seguente, tenendo conto che il programma '**`makeboot`**' si presume collocato in una directory prevista tra i vari percorsi della variabile di ambiente '**`PATH`**':

```
# makeboot minix /dev/fd0 [Invio]

  Wrote sector 0
  Wrote sector 1
```

Se il programma si accorge che il settore di avvio del dischetto contiene già qualcosa, si rifiuta di procedere, a meno di usare l'opzione '**`-f`**':

```
# makeboot -f minix /dev/fd0 [Invio]

  Boot block isn't empty, zap it first
  Wrote sector 0
  Wrote sector 1
```

È importante sapere che questo programma di avvio colloca in memoria il programma o il kernel da avviare a partire dall'indirizzo efficace 10000_{16}. Pertanto, dopo l'avvio effettivo, rimane inutilizzato lo spazio di memoria da 00500_{16} $0FFFF_{16}$.

Riferimenti

- Andrew S. Tanenbaum, *Operating Systems: Design and Implementation*, **prima edizione**, 1987, Prentice-Hall, ISBN 0-13-637406-9
  Appendice B: *introduction to the IBM PC*
- MAD, *Assembly tutorial*
  http://www.xs4all.nl/~smit/asm01001.htm
- Wikipedia, *x86 instruction listings*
  http://en.wikipedia.org/wiki/X86_instruction_listings
- *The x86 Interrupt List, aka "Ralf Brown's Interrupt List", "RBIL"*
  http://www.cs.cmu.edu/~ralf/files.html
- *Computer interrupt*
  http://wayback.archive.org/web/20040101000000/http://calab.kaist.ac.kr/~hyoon/courses/cs310_2001fa01ll/micro17.ppt*
  http://www.ece.msstate.edu/~reese/EE3724/lectures/interrupt/interrupt.pdf
- BiosCentral, *BIOS data area*
  http://www.bioscentral.com/misc/bda.htm
- Robert de Bath, *Linux 8086 development environment*
  http://homepage.ntlworld.com/robert.debath/
  http://homepage.ntlworld.com/robert.debath/dev86/

[1] **Bcc, As86, Ld86** GNU GPL

[2] Va specificato che si tratta del compilatore Bcc di Bruce Evans, perché con questo nome o con questa sigla si trovano più facilmente riferimenti a compilatori C diversi, per esempio quello di Borland.

[3] **Bootblocks** GNU GPL

Parte xxxiv

Studio per un sistema a 16 bit

«

Introduzione a os16 1321

Organizzazione 1321
Le directory 1322
La struttura degli eseguibili 1322
Caricamento del kernel 1323
Informazioni diagnostiche 1323
Tabelle 1324
Guida di stile 1324
Tipi derivati speciali 1326

Caricamento ed esecuzione del kernel 1327

Dal file su disco alla copia in memoria 1327
File «kernel/main/crt0.s» 1328
File «kernel/main.h» e «kernel/main/∗» 1330

Funzioni interne legate all'hardware 1333

Libreria: «lib/sys/os16.h» e «lib/sys/os16/...» 1333
Funzioni di basso livello dei file «kernel/ibm_i86/∗» 1334
Gestione della console 1336
Gestione dei dischi 1338

Gestione della memoria 1341

File «kernel/memory.h» e «kernel/memory/...» 1341
Scansione della mappa di memoria 1343

Gestione dei terminali virtuali 1345

Dispositivi 1347

File «lib/sys/os16.h» e directory «lib/sys/os16/» 1347
File «kernel/devices.h» e «kernel/devices/...» 1347
Numero primario e numero secondario 1348
Dispositivi previsti 1349

Gestione del file system 1351

File «kernel/fs/sb_...» 1351
File «kernel/fs/zone_...» 1353
File «kernel/fs/inode_...» 1354
Fasi dell'innesto di un file system 1357
File «kernel/fs/file_...» 1358
Descrittori di file 1359
File «kernel/fs/path_...» 1360
File «kernel/fs/fd_...» 1362

Gestione dei processi 1365

File «kernel/proc/_isr.s» e «kernel/proc/_ivt_load.s» . . . 1365
La tabella dei processi 1370
Chiamate di sistema 1373
File «kernel/proc/...» 1374

Caricamento ed esecuzione delle applicazioni 1377

Caricamento in memoria 1377
Il codice iniziale dell'applicativo 1378

Organizzazione 1321
Le directory 1322
La struttura degli eseguibili 1322
Caricamento del kernel 1323
Informazioni diagnostiche 1323
Tabelle 1324
Guida di stile 1324
Tipi derivati speciali 1326

`addr_t` 1326 `diag.h` 1323 `directory_t` 1326 `dsk_chs_t` 1326 `dsk_t` 1326 `fd_t` 1326 `file_t` 1326 `inode_t` 1326 `memory_t` 1326 `offset_t` 1326 `sb_t` 1326 `segment_t` 1326 `tty_t` 1326 `zno_t` 1326

os16 è uno studio che applica qualche rudimento relativo ai sistemi operativi, basandosi sull'architettura x86-16 del vecchio IBM PC, utilizzando come strumenti di sviluppo Bcc, As86 e Ld86 (oltre a GNU GCC per controllare meglio la sintassi del codice C), su un sistema GNU/Linux. Il risultato non è un sistema operativo utilizzabile, ma una struttura su cui poter fare esperimenti e di cui è possibile mostrare (in termini tipografici) ed eventualmente descrivere ogni riga di codice.

os16 contiene uno schedulatore banale e molto limitato, un'organizzazione dei processi ad albero e una funzionalità limitata di amministrazione dei segnali, una gestione del file system Minix 1 (ma di unità intere, senza partizioni), una shell banale e qualche programma di servizio di esempio.

Per poter giungere rapidamente a un risultato e comunque per semplificare il codice, os16 utilizza le funzioni del BIOS tradizionale, le quali hanno lo svantaggio di impegnare in modo esclusivo l'elaboratore nel momento del loro funzionamento (dato che non possono essere rientranti). È noto che un sistema operativo multiprogrammato dignitoso non può avvalersi di tali funzionalità; pertanto, se si vuole studiare os16, non va dimenticato questo principio, benché qui sia stato trascurato.

Il kernel di os16 è monolitico, nel senso che incorpora tutte le proprie funzioni in un solo programma. Purtroppo, la dimensione del codice del kernel (e di qualunque altro processo di os16) non può superare i 64 Kibyte, ma la sua dimensione è già molto vicina a tale valore. Pertanto, non è possibile aggiungere funzionalità a questo sistema. Si può osservare che anche ELKS (*http://elks.sourceforge.net/*) soffra dello stesso limite e, d'altra parte, si può apprezzare come Minix 2 (*http://minix1.woodhull.com/*) riesca a superarlo attraverso un'organizzazione a micro-kernel.

Organizzazione

Tutti i file di os16 dovrebbero essere disponibili a partire da *allegati/os16*. In particolare i file '`floppy.a`' e '`floppy.b`' sono le immagini di due dischetti da 1440 Kibyte, contenenti un file system Minix 1: il primo predisposto attraverso Bootblocks (sezione u0.5) per l'avvio di un kernel denominato '`boot`'; il secondo usato per essere innestato nella directory '`/usr/`' del primo.

Gli script preparati per lo sviluppo di os16 prevedono che i file-immagine dei dischetti vadano innestati nelle directory '`/mnt/os16.a/`' e '`/mnt/os16.b/`'. Pertanto, se si ricompila os16, tali directory vanno predisposte (oppure vanno modificati gli script con l'organizzazione che si preferisce attuare).

Per la verifica del funzionamento del sistema, è previsto l'uso equivalente di Bochs o di Qemu. Per questo scopo sono disponibili gli script '`bochs`' e '`qemu`' (rispettivamente i listati i160.1.1 e i160.1.2), con le opzioni necessarie a operare correttamente.

Per la compilazione del lavoro si usa lo script ‘`makeit`’ (listato i160.1.3), il quale ricrea ogni volta i file-make, basandosi sui file presenti effettivamente nelle varie directory previste. Questo script, alla fine della compilazione, copia il kernel nel file-immagine del primo dischetto (purché risulti innestato come previsto nella directory ‘`/mnt/os16.a/`’) e con esso copia anche gli applicativi principali, mentre il resto viene copiato nel secondo.

Nello script ‘`makeit`’, la variabile di ambiente ‘**TAB**’ deve contenere esattamente il carattere di tabulazione (<*HT*>), corrispondente al codice 09_{16}. Il file che viene distribuito contiene invece l’assegnamento di uno spazio puro e semplice, che va modificato a mano, sostituendolo con tale codice. La riga da modificare è quella in cui la variabile viene dichiarata:

```
local TAB=" "
```

Va osservato che il lavoro si basa su un file system Minix 1 (sezione 68.7) perché è molto semplice, ma soprattutto, la prima versione è quella che può essere utilizzata facilmente in un sistema operativo GNU/Linux (sul quale avviene lo sviluppo di os16). È bene sottolineare che si tratta della versione con nomi da 14 caratteri, ovvero quella tradizionale del sistema operativo Minix, mentre nei sistemi GNU/Linux, la creazione predefinita di un file system del genere produce una versione particolare, con nomi da 30 caratteri.

Le directory

Gli script descritti nella sezione precedente, si trovano all’inizio della gerarchia prevista per os16. Le directory successive dividono in modo molto semplice le varie componenti per la compilazione:

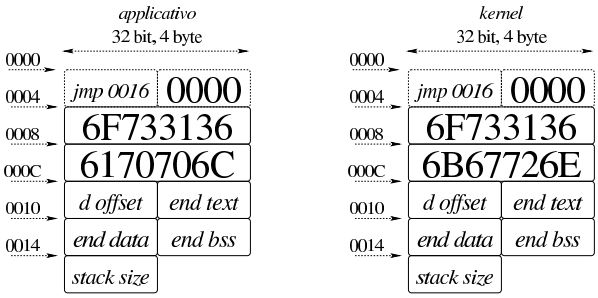
Directory	Contenuto
‘`applic/`’	File delle applicazioni da usare con os16.
‘`kernel/`’	File per la realizzazione del kernel, inclusi i file di intestazione specifici.
‘`lib/`’	File di intestazione generali, file della libreria C per le applicazioni e, per quanto possibile, anche per il kernel.
‘`ported/`’	Applicativi di altri autori, adattati per os16.
‘`skel/`’	Scheletro del file system complessivo, con i file di configurazione e le pagine di manuale.

La libreria C non è completa, limitandosi a contenere ciò che serve per lo stato di avanzamento attuale del lavoro. Si osservi che nella directory ‘`lib/bcc/`’ si collocano file contenenti una libreria di funzioni in linguaggio assemblatore, necessaria al compilatore Bcc per compiere il proprio lavoro correttamente con valori da 32 bit. Tale libreria è scritta dall’autore originale di Bcc, Bruce Evans, e viene inclusa in modo da garantire che la compilazione non richieda alcun file estraneo.

La struttura degli eseguibili

Nell’ottica della massima semplicità, gli eseguibili di os16 hanno un’intestazione propria, schematizzata dalla figura successiva. Tale intestazione viene ottenuta attraverso il file ‘`crt0.s`’, che è comunque differente se si tratta di un applicativo o del kernel.

Figura u142.2. Struttura iniziale dei file eseguibili di os16.



Nella figura si mettono a confronto la parte iniziale dell’eseguibile di un applicativo con quella del kernel di os16. Nei primi quattro byte c’è un’istruzione di salto al codice che si trova subito dopo l’intestazione, quindi appare un’impronta di riconoscimento che occupa quattro byte. Tale impronta è la rappresentazione esadecimale della stringa «os16». Successivamente appare un’altra impronta, con cui si distingue se si tratta di un applicativo o del kernel; si tratta in pratica della sequenza di «appl», oppure di «kern». Tuttavia, a causa dell’inversione dell’ordine dei byte, in pratica, se si visualizza il file binario si legge «61so», «lppa» e «nrek».

Dopo l’impronta di riconoscimento si trovano, rispettivamente, lo scostamento del segmento dati, espresso in multipli di 16 (in pratica, 1234_{16} rappresenterebbe uno scostamento di 12340_{16} byte, rispetto all’inizio del codice), gli indirizzi conclusivi dell’area del codice, dei dati inizializzati e di quelli non inizializzati. Alla fine viene indicata la dimensione richiesta per la pila dei dati. Ciò che appare dopo è il codice del programma.

Il kernel e gli applicativi di os16 sono compilati in modo da rendere indipendenti l’area del codice rispetto a quella dei dati (si dice che hanno aree «I&D separate»).

Nel caso del kernel, le informazioni successive alle impronte di riconoscimento non vengono utilizzate, perché il kernel viene collocato in uno spazio preciso in memoria: l’area dati va a trovarsi dall’indirizzo efficace 00500_{16} fino a $104FF_{16}$ incluso, mentre l’area del codice inizia da 10500_{16} fino alla fine.

Per fare in modo che il proprio sistema GNU possa riconoscere correttamente questi file, si può modificare la configurazione del file ‘`/etc/magic`’, aggiungendo le righe seguenti:

```
4       quad        0x6B65726E6F733136      os16 kernel
4       quad        0x6170706C6F733136      os16 application
```

Caricamento del kernel

Il kernel è preparato per trovarsi inizialmente in memoria, tale e quale al file da cui viene caricato, a partire dall’indirizzo efficace 10000_{16}, così come avviene quando si utilizza Bootblocks (a cui si è già accennato nel capitolo). Successivamente il kernel stesso si sposta, copiandosi inizialmente a partire dall’indirizzo efficace 30000_{16}, quindi suddividendosi e mettendo la propria area dati a partire dall’indirizzo 00500_{16} e l’area codice da 10500_{16} (lo spazio di memoria che va da 00000_{16} a $004FF_{16}$ incluso, non può essere utilizzato, perché contiene la tabella IVT e l’area BDA, secondo l’architettura degli elaboratori IBM PC tradizionali).

Informazioni diagnostiche

Nel codice del kernel vengono usate, in varie occasioni, delle funzioni che hanno lo scopo di visualizzare delle informazioni diagnostiche. Queste funzioni sono raccolte in file della directory ‘`kernel/diag/`’, a cui si abbina il file di intestazione ‘`kernel/diag.h`’ (listato u0.3 e successivi). In generale, in questa documentazione, non viene dato molto spazio alla descrizione di queste funzioni, perché hanno un ruolo marginale e sono fatte per essere modificate in base alle esigenze di verifica del momento.

Tabelle

«

Nel codice del kernel si utilizzano spesso delle informazioni organizzate in memoria in forma di tabella. Si tratta precisamente di array, le cui celle sono costituite generalmente da variabili strutturate. Queste tabelle, ovvero gli array che le rappresentano, sono dichiarate come variabili pubbliche; tuttavia, per facilitare l'accesso ai rispettivi elementi e per uniformità di comportamento, viene abbinata loro una funzione, con un nome terminante per '`…_reference()`', con cui si ottiene il puntatore a un certo elemento della tabella, fornendo gli argomenti appropriati. Per esempio, la tabella degli inode in corso di utilizzazione viene dichiarata così nel file '`kernel/inode/inode_table.c`':

```
inode_t inode_table[INODE_MAX_SLOTS];
```

Successivamente, la funzione ***inode_reference()*** offre il puntatore a un certo inode:

```
inode_t *inode_reference (dev_t device, ino_t ino);
```

Guida di stile

«

Per cercare di dare un po' di uniformità al codice del kernel e a quello della libreria, dove possibile, i nomi delle variabili seguono una certa logica, riassunta dalla tabella successiva.

Tipo	Nome	Utilizzo
`inode_t *`	`inode` `inode_…`	Puntatore a un inode (puntatore a un elemento della tabella di inode).
`ino_t`	`ino` `ino_…`	Numero di inode, nell'ambito di un certo super blocco (ammesso che sia abbinato effettivamente a un dispositivo).
`int`	`fdn` `fdn_…`	Numero del descrittore di un file (indice all'interno della tabella dei descrittori).
`fd_t *`	`fd` `fd_…`	Puntatore a un descrittore di file (puntatore a un elemento della tabella di descrittori).
`int`	`fno` `fno_…`	Numero del file di sistema (indice all'interno della tabella dei file di sistema).
`zno_t`	`zone` `zone_…`	Numero assoluto di una «zona» del file system Minix.
`zno_t`	`fzone` `fzone_…`	Numero relativo di una «zona» del file system Minix. In questo caso, il numero della zona è relativo al file, dove la prima zona del file ha il numero zero.

`off_t`	`offset`
	`offset_…`
	`off_…`
		Scostamento, secondo il significato del tipo derivato '**`off_t`**'.

`size_t` `ssize_t`	`size` `size_…`	Dimensione, secondo il significato dei tipi derivati '**`size_t`**' o '**`ssize_t`**'.
`size_t` `ssize_t`	`count` `count_…`	Quantità, quando il tipo '**`size_t`**' è appropriato.
`blkcnt_t`	`blkcnt` `blkcnt_…`	Quantità espressa in blocchi del file system (in questo caso, trattandosi di un file system Minix 1, si intendono zone).

Tipo	Nome	Utilizzo
`blksize_t`	`blksize` `blksize_…`	Dimensione del blocco del file system, espressa in byte (in questo caso, trattandosi di un file system Minix 1, si intende la dimensione della zona).
`int`	`fno` `fno_…`	Numero di file system.
`int`	`oflags` `oflags_…`	Opzioni relative all'apertura di un file, annotate nella tabella dei file di sistema: indicatori di sistema.
`int`	`status` `status_…`	Valore intero restituito da una funzione, quando la risposta contiene solo l'indicazione di un successo o di un insuccesso.
`void *`	`pstatus`	Puntatore restituito da una funzione, quando interessa sapere solo se si tratta di un esito valido.
`char *`	`path` `path_…`	Percorso del file system.
`dev_t`	`device` `device_…`	Numero di dispositivo, contenente sia il numero primario, sia quello secondario (*major*, *minor*).
`int`	`n` `n_…`	Dimensione di qualcosa, di tipo '**`int`**'.
`char *`	`string` `string_…`	Area di memoria da considerare come stringa.
`void *`	`buffer` `buffer_…`	Area di memoria destinata ad accogliere un'informazione di tipo imprecisato.
`int`	`n` `n_…`	Dimensione o quantità di qualcosa, espressa attraverso il tipo '**`int`**'.
`int`	`c` `c_…`	Un carattere senza segno trasformato nel tipo '**`int`**'.
`struct stat`	`st` `st_…`	Variabile strutturata usata per rappresentare lo stato di un file, secondo il tipo '**`struct stat`**'.
`FILE *`	`fp` `fp_…`	Puntatore che rappresenta un flusso di file.
`DIR *`	`dp` `dp_…`	Puntatore che rappresenta un flusso relativo a una directory.
`struct dirent`	`dir` `dir_…`	Variabile strutturata contenente le informazioni su una voce di una directory.
`struct password`	`pws` `pws_…`	Variabile strutturata contenente le informazioni di una voce del file '`/etc/passwd`'.
`struct tm`	`tms` `tms_…`	Variabile strutturata contenente le componenti di un orario.

Tipo	Nome	Utilizzo
`struct tm *`	`timeptr` `timeptr_…`	Puntatore a una variabile strutturata contenente le componenti di un orario.

Tipi derivati speciali

Nel codice del kernel si usano dei tipi derivati speciali, riassunti nella tabella successiva.

File di intestazione	Tipo speciale	Descrizione
‘`kernel/memory.h`’	`addr_t`	Variabile scalare, in grado di rappresentare un indirizzo efficace di memoria (un indirizzo che vada da 00000_{16} a $FFFFF_{16}$).
‘`kernel/memory.h`’	`segment_t`	Variabile scalare, a 16 bit, usata per rappresentare il valore di un registro di segmento.
‘`kernel/memory.h`’	`offset_t`	Variabile scalare, a 16 bit, usata per rappresentare lo scostamento di memoria, a partire dall’inizio di un segmento. Questo tipo di variabile non va confuso con il tipo ‘**`off_t`**’, il quale è un valore con segno e di rango maggiore rispetto a questo ‘**`offset_t`**’.
‘`kernel/memory.h`’	`memory_t`	Variabile strutturata, adatta a contenere tutte le coordinate utili a individuare una certa area di memoria, secondo l’architettura prevista da os16.
‘`kernel/tty.h`’	`tty_t`	Variabile strutturata, adatta a contenere le informazioni e lo stato di un terminale.
‘`kernel/ibm_i86.h`’	`dsk_t`	Variabile strutturata, adatta a contenere le informazioni hardware di una certa unità di memorizzazione.
‘`kernel/ibm_i86.h`’	`dsk_chs_t`	Variabile strutturata, adatta a contenere le coordinate di un certo settore (cilindro, testina e settore) in un’unità di memorizzazione.
‘`kernel/fs.h`’	`zno_t`	Variabile scalare, per rappresentare un numero di una zona, secondo la terminologia del file system Minix.
‘`kernel/fs.h`’	`sb_t`	Variabile strutturata, adatta a contenere tutte le informazioni di un super blocco, relativo a un dispositivo di memorizzazione innestato.
‘`kernel/fs.h`’	`inode_t`	Variabile strutturata, adatta a contenere tutte le informazioni di un inode aperto nel sistema.
‘`kernel/fs.h`’	`file_t`	Variabile strutturata, adatta a contenere i dati di un file di sistema.
‘`kernel/fs.h`’	`fd_t`	Variabile strutturata, adatta a contenere i dati di un descrittore di file, ovvero del file di un certo processo elaborativo.
‘`kernel/fs.h`’	`directory_t`	Variabile strutturata, adatta a contenere una voce di una directory.

Caricamento ed esecuzione del kernel

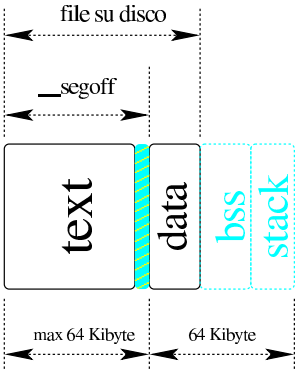
Dal file su disco alla copia in memoria 1327
File «kernel/main/crt0.s» 1328
File «kernel/main.h» e «kernel/main/*» 1330

`crt0.s` 1328 `main()` 1330

Il kernel di os16 (ma così vale anche per gli applicativi) viene compilato senza un’intestazione predefinita, pertanto questa viene costruita nel primo file: ‘`crt0.s`’. Questo file ha lo scopo di eseguire la funzione ***main()*** del kernel, in cui si sintetizza il funzionamento dello stesso.

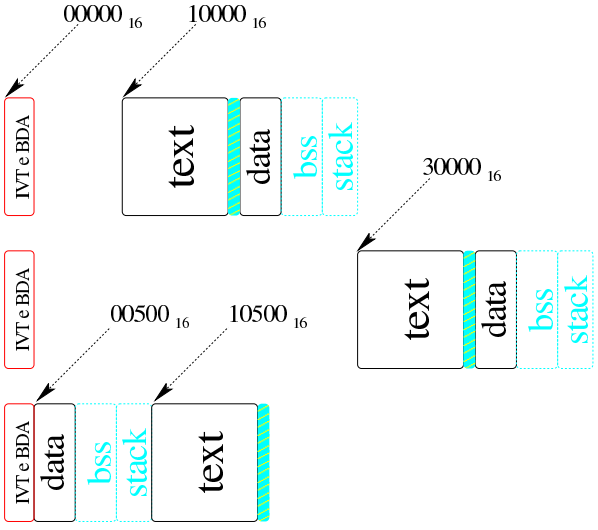
Dal file su disco alla copia in memoria

Il file del kernel prodotto dagli strumenti di sviluppo è strutturato come sintetizza il disegno seguente:



La prima parte del file è utilizzata dal codice (*text*), quindi ci può essere un piccolissimo spazio inutilizzato, seguito dalla porzione che riguarda i dati, tenendo conto che nel file ci sono solo i dati inizializzati, mentre gli altri non hanno bisogno di essere rappresentati, ma in memoria occupano comunque il loro spazio.

Il kernel è organizzato per tenere separate l’area delle istruzioni da quella dei dati, pertanto il compilatore (precisamente il «collegatore», ovvero il *linker*) offre il simbolo ***__segoff***, con il quale si conosce la distanza del segmento dei dati dall’inizio del file. Il valore di questo scostamento è espresso in «paragrafi», ovvero in multipli di 16; in pratica si tratta dello scostamento da utilizzare in un registro di segmento. Dal momento che lo scostamento effettivo è costituito dalla dimensione dell’area del codice, approssimata per eccesso ai 16 byte successivi, tra la fine dell’area codice e l’inizio di quella dei dati c’è quel piccolo spazio vuoto a cui già si è fatto riferimento.



Il kernel viene caricato in memoria, con l'ausilio di Bootblocks, all'indirizzo 10000_{16}. Da lì il kernel si mette in funzione e, prima si copia all'indirizzo 30000_{16}, quindi riprende a funzionare dal nuovo indirizzo, poi si copia mettendo i dati a partire dall'indirizzo 00500_{16} (dopo la tabella IVT e dopo l'area BDA) e il codice a partire dall'indirizzo 10500_{16}. Alla fine, riprende a funzionare dall'indirizzo 10500_{16}. La pila dei dati (*stack*) viene attivata solo quando il kernel ha trovato la sua collocazione definitiva.

File «kernel/main/crt0.s»

Listato i160.7.2.

Dopo il preambolo in cui si dichiarano i simboli esterni e quelli interni da rendere pubblici, con l'istruzione '**`entry startup`**' si dichiara all'assemblatore che il punto di partenza è costituito dal simbolo '**`startup`**', ma in ogni caso questo deve essere all'inizio del codice, mancando un'intestazione precostituita. In pratica, la primissima cosa che si ottiene nel file eseguibile finale è un'istruzione di salto a una posizione più avanzata del codice, dove si colloca il simbolo '**`startup_code`**', e nello spazio intermedio (tra quell'istruzione di salto e il codice che si trova a partire da '**`startup_code`**') si collocano le impronte di riconoscimento, oltre ai dati sulla dislocazione dell'eseguibile in memoria.

```
…
entry startup
…
startup:
    jmp startup_code
…
startup_code:
…
```

Tra la prima istruzione di salto e le impronte di riconoscimento, introdotte dal simbolo '**`magic`**', c'è uno spazio vuoto (nullo), calcolato automaticamente in modo da garantire che la prima impronta inizi all'indirizzo relativo 0004_{16}. Di seguito vengono gli altri dati.

```
…
startup:
    jmp startup_code
filler:
    .space (0x0004 - (filler - startup))
magic:
    .data4 0x6F733136
    .data4 0x6B65726E
segoff:
    .data2 __segoff
etext:
    .data2 __etext
edata:
    .data2 __edata
ebss:
    .data2 __end
stack_size:
    .data2 0x0000
.align 2
startup_code:
…
```

A partire da '**`startup_code`**' viene analizzato il valore effettivo del registro ***CS***. Se questo è pari a 1000_{16}, significa che il kernel si trova in memoria a partire dall'indirizzo efficace 10000_{16}, ma in tal caso si salta a una procedura che copia il kernel in un'altra posizione di memoria (30000_{16}); se invece il valore di ***CS*** viene riconosciuto pari a quello della destinazione della prima copia, si passa a un'altra procedura che scompone l'area dati e l'area codice (testo) del kernel, in modo da collocare l'area dati a partire da 00500_{16} e l'area codice a partire da 10500_{16}. Quando si riconosce che il valore di ***CS*** è quello finale, si salta al simbolo '**`main_code`**' e da lì inizia il lavoro vero e proprio.

```
…
startup_code:
    mov cx, cs
    xor cx, #0x1000
    jcxz move_code_from_0x1000_to_0x3000
    mov cx, cs
    xor cx, #0x3000
    jcxz move_code_from_0x3000_to_0x0050
    mov cx, cs
    xor cx, #0x1050
    jcxz main_code
    hlt
    jmp startup_code
move_code_from_0x1000_to_0x3000:
    …
    jmp far #0x3000:#0x0000
move_code_from_0x3000_to_0x0050:
    …
    jmp far #0x1050:#0x0000
main_code:
…
```

Non si prevede che il kernel possa trovarsi in memoria in una collocazione differente da quelle stabilite nelle varie fasi di avvio, pertanto, in caso contrario, si crea semplicemente un circolo vizioso senza uscita.

Dal simbolo '**`main_code`**' inizia finalmente il lavoro e si procede con l'allineamento dei registri dei segmenti dei dati, in modo che siano tutti corrispondenti al valore previsto: 0050_{16} (il segmento in cui inizia l'area dati, secondo la collocazione prevista). Viene poi posizionato il valore del registro ***SP*** a zero, in modo che al primo inserimento questo punti esattamente all'indirizzo più grande che si possa raggiungere nel segmento dati ($FFFE_{16}$, considerato che gli inserimenti nella pila sono a 16 bit).

```
…
main_code:
    mov  ax, #0x0050
    mov  ds, ax
    mov  ss, ax
    mov  es, ax
    mov sp, #0x0000
    …
```

Appena la pila diventa operativa, si inizializza anche il registro ***FLAGS***, verificando di disabilitare inizialmente le interruzioni.

```
…
main_code:
    …
    push #0
    popf
    cli
    …
```

A questo punto, si chiama la funzione ***main()***, fornendo come argomenti tre valori a zero.

```
…
main_code:
    …
    push #0
    push #0
    push #0
    call _main
    add  sp, #2
    add  sp, #2
    add  sp, #2
…
```

Nel caso la funzione dovesse terminare e restituire il controllo, si passerebbe al codice successivo al simbolo '**`halt`**', con cui si crea un ciclo senza uscita, corrispondente alla conclusione del funzionamento del kernel.

```
…
halt:
    hlt
    jmp halt
…
```

Utilizzando il compilatore Bcc per compilare ciò che descrive la funzione ***main()***, viene richiesta la presenza della funzione ***__mkargv()***

(il simbolo '**___mkargv**'), che in questo caso può limitarsi a non fare alcunché.

```
…
___mkargv:
    ret
…
```

File «kernel/main.h» e «kernel/main/*»

« Listato u0.7 e successivi.

Tutto il lavoro del kernel di os16 si sintetizza nella funzione ***main()***, contenuta nel file '`kernel/main/main.c`'. Per poter dare un significato a ciò che vi appare al suo interno, occorre conoscere tutto il resto del codice, ma inizialmente è utile avere un'idea di ciò che succede, se poi si vuole compilare ed eseguire il sistema operativo.

La funzione ***main()*** viene dichiarata secondo la forma tradizionale di un programma per sistemi POSIX, ma gli argomenti che riceve dalla chiamata contenuta nel file '`kernel/main/crt0.s`' sono nulli, perché nessuna informazione gli viene passata effettivamente.

```
…
int
main (int argc, char *argv[], char *envp[])
{
    …
    tty_init ();
    k_printf ("os16 build %s ram %i Kibyte\n", BUILD_DATE,
              int12 ());
    dsk_setup ();
    heap_clear ();
    proc_init ();
    menu ();
    …
```

Dopo la dichiarazione delle variabili si inizializza la gestione del video della console con la funzione ***tty_init()***, si mostra un messaggio iniziale, quindi si passa alla predisposizione di ciò che serve, prima di poter avviare dei processi. In particolare va osservata la funzione ***heap_clear()***, la quale inizializza con il codice $FFFF_{16}$ lo spazio di memoria libero, tra la fine delle variabili «statiche» e il livello che ha raggiunto in quel momento la pila dei dati. Successivamente, avendo marcato in questo modo quello spazio, diventa possibile riconoscere empiricamente quanto spazio di quella porzione di memoria avrebbe potuto essere utilizzato, senza essere sovrascritto dalla pila dei dati. Il messaggio iniziale contiene la data di compilazione e la memoria libera (la macro-variabile ***BUILD_DATE*** viene definita dallo script '**makeit**', usato per la compilazione, creando il file '`kernel/main/build.h`' che viene poi incluso dal file '`kernel/main/main.c`').

L'attivazione della gestione dei processi (e delle interruzioni) con la funzione ***proc_init()***, comporta anche l'innesto del file system principale (chiamando da lì la funzione ***sb_mount()***).

```
…
int
main (int argc, char *argv[], char *envp[])
{
    …
    menu ();
    for (exit = 0; exit == 0;)
      {
        sys (SYS_0, NULL, 0);
        dev_io ((pid_t) 0, DEV_TTY, DEV_READ, 0L, &key, 1,
                NULL);
        …
        switch (key)
          {
            case 'h':
                menu ();
                break;
            …
            case 'x':
                exit = 1;
                break;
            case 'q':
                k_printf ("System halted!\n");
                return (0);
                break;
          }
      }
    }
…
```

A questo punto il kernel ha concluso le sue attività preliminari e, per motivi diagnostici, mostra un menù, quindi inizia un ciclo in cui ogni volta esegue una chiamata di sistema nulla e poi legge un carattere dalla tastiera: se risulta premuto un tasto previsto, fa quanto richiesto e riprende il ciclo. La chiamata di sistema nulla serve a far sì che lo schedulatore ceda il controllo a un altro processo, ammesso che questo esista, consentendo l'avvio di processi ancor prima di avere messo in funzione quel processo che deve svolgere il ruolo di '**init**'.

In generale le chiamate di sistema sono fatte per essere usate solo dalle applicazioni; tuttavia, in pochi casi speciali il kernel le deve utilizzare come se fosse proprio un'applicazione. Qui si rende necessario l'uso della chiamata nulla, perché quando è in funzione il codice del kernel non ci possono essere interruzioni esterne e quindi nessun altro processo verrebbe messo in condizione di funzionare.

Le funzioni principali disponibili in questa modalità diagnostica sono riassunte nella tabella successiva:

Tasto	Risultato
$[h]$	Mostra il menù di funzioni disponibili.
$[1]$	Invia il segnale '**SIGKILL**' al processo numero uno.
$[2]$...$[9]$	Invia il segnale '**SIGTERM**' al processo con il numero corrispondente.
$[A]$...$[F]$	Invia il segnale '**SIGTERM**' al processo con il numero da 10 a 15.
$[a]$, $[b]$, $[c]$	Avvia il programma '`/bin/aaa`', '`/bin/bbb`' o '`/bin/ccc`'.
$[f]$	Mostra l'elenco dei file aperti nel sistema.
$[m]$, $[M]$	Innesta o stacca il secondo dischetto dalla directory '`/usr/`'.
$[n]$, $[N]$	Mostra l'elenco degli inode aperti: l'elenco è composto da due parti.
$[l]$	Invia il segnale '**SIGCHLD**' al processo numero uno.
$[p]$	Mostra la situazione dei processi e altre informazioni.
$[x]$	Termina il ciclo e successivamente si passa all'avvio di '`/bin/init`'.
$[q]$	Ferma il sistema.

Premendo $[x]$, il ciclo termina e il kernel avvia '`/bin/init`'. Quindi si mette in un altro ciclo, dove si limita a passare ogni volta il controllo allo schedulatore, attraverso la chiamata di sistema nulla.

```
…
int
main (int argc, char *argv[], char *envp[])
{
    …
    menu ();
    for (exit = 0; exit == 0;)
      {
        …
      }
    exec_argv[0] = "/bin/init";
    exec_argv[1] = NULL;
    pid = run ("/bin/init", exec_argv, NULL);
    while (1)
      {
        sys (SYS_0, NULL, 0);
      }
    …
}
```

Figura u143.15. Aspetto di os16 in funzione, con il menù in evidenza.

```
os16 build 20YY.MM.DD HH:MM:SS ram 639 Kibyte
.-------------------------------------------------------------.
| [h]      show this menu                                      |
| [p]      process status and memory map                       |
| [1]..[9] kill process  1 to 9                                |
| [A]..[F] kill process 10 to 15                               |
| [l]      send SIGCHLD to process 1                           |
| [a]..[c] run programs '/bin/aaa' to '/bin/ccc' in parallel   |
| [f]      system file status                                  |
| [n], [N] list of active inodes                               |
| [m], [M] mount/umount '/dev/dsk1' at '/usr/'                 |
| [x]      exit interaction with kernel and start '/bin/init'  |
| [q]      quit kernel                                         |
'-------------------------------------------------------------'
```

Figura u143.16. Aspetto di os16 in funzione mentre visualizza anche la tabella dei processi avviati (tasto [*p*]).

```
ababaaababaaababaabbaaababaaababap
pp  p pg
id id rp  tty  uid euid suid usage s iaddr isiz daddr dsiz sp   name
 0  0  0 0000    0    0    0 00.35 R 10500 eb7c 00500 0000 ffc8 os16 kernel
 0  1  0 0000    0    0    0 00.33 r 2f100 0600 2f700 aa00 a8e8 /bin/ccc
 0  2  0 0000    0    0    0 00.01 r 1f100 0600 84300 aa00 a8e8 /bin/ccc
 2  3  0 0000   10   10   10 00.01 r 1f100 0600 44b00 aa00 a8e8 /bin/ccc
 0  4  0 0000    0    0    0 00.17 r 21600 0600 3a100 aa00 a8e8 /bin/ccc
 4  5  0 0000   10   10   10 00.02 r 21c00 2400 6f100 a900 a86c /bin/aaa
 4  6  0 0000   11   11   11 00.02 s 24000 2500 59d00 a900 a8b6 /bin/bbb
 0  7  0 0000    0    0    0 00.13 r 26500 0600 64600 aa00 a8e8 /bin/ccc
 7  8  0 0000   10   10   10 00.02 r 26b00 2400 8ee00 a900 981e /bin/aaa
 7  9  0 0000   11   11   11 00.02 s 2bf00 2500 79a00 a900 a8b6 /bin/bbb
CS=1050 DS=0050 SS=0050 ES=0050 BP=ffe4 SP=ffe4 heap_min=878c etext=eb7c edata=1
b3c ebss=4c34 ksp=ffc8 clock=0000084b, time elapsed=00:01:57
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffe00000003ffffffffffffffffffffffffff
fffe000000000001fffffffff0007fffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff800
0000000000000000000000000000000000000007ffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffff7fffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffbfffffffffffffffffffffffffffffffffffffffffe0000000000000000000000000f

abbaaaaababaaababaaabbaabbaababab baabbbbbbbbbbbb
```

Figura u143.17. Aspetto di os16 in funzione con il menù in evidenza, dopo aver premuto il tasto [*x*] per avviare '**init**'.

```
os16 build 20YY.MM.DD HH:MM:SS ram 639 Kibyte
.-----------------------------------------------------------------.
| [h]      show this menu                                          |
| [p]      process status and memory map                           |
| [1]..[9] kill process  1 to 9                                    |
| [A]..[F] kill process 10 to 15                                   |
| [l]      send SIGCHLD to process 1                               |
| [a]..[c] run programs '/bin/aaa' to '/bin/ccc' in parallel       |
| [f]      system file status                                      |
| [n], [N] list of active inodes                                   |
| [m], [M] mount/umount '/dev/dsk1' at '/usr/'                     |
| [x]      exit interaction with kernel and start '/bin/init'      |
| [q]      quit kernel                                             |
'-----------------------------------------------------------------'
init
os16: a basic os. [Ctrl q], [Ctrl r], [Ctrl s], [Ctrl t] to change
console.
This is terminal /dev/console0
Log in as "root" or "user" with password "ciao" :-)
login:
```


Funzioni interne legate all'hardware

Libreria: «lib/sys/os16.h» e «lib/sys/os16/...» 1333

Funzioni di basso livello dei file «kernel/ibm_i86/*» 1334

Gestione della console 1336

Gestione dei dischi 1338

`bp()` 1333 `cli()` 1334 `con_char_read()` 1337 `con_char_ready()` 1337 `con_char_wait()` 1337 `con_init()` 1337 `con_putc()` 1337 `con_scroll()` 1337 `con_select()` 1337 `cs()` 1333 `ds()` 1333 `dsk_chs_t` 1338 `dsk_read_bytes()` 1338 `dsk_read_sectors()` 1338 `dsk_reset()` 1338 `dsk_sector_to_chs()` 1338 `dsk_setup()` 1338 `dsk_t` 1338 `dsk_write_bytes()` 1338 `dsk_write_sectors()` 1338 `es()` 1333 `ibm_i86.h` 1333 `int10_00()` 1334 `int10_02()` 1334 `int10_05()` 1334 `int12()` 1334 `int13_00()` 1334 `int13_02()` 1334 `int13_03()` 1334 `int16_00()` 1334 `int16_01()` 1334 `int16_02()` 1334 `in_16()` 1334 `in_8()` 1334 `irq_off()` 1334 `irq_on()` 1334 `os16.h` 1333 `out_16()` 1334 `out_8()` 1334 `ram_copy()` 1334 `seg_d()` 1333 `seg_i()` 1333 `sp()` 1333 `ss()` 1333 `sti()` 1334 `_bp()` 1333 `_cs()` 1333 `_ds()` 1333 `_es()` 1333 `_int10_00()` 1334 `_int10_02()` 1334 `_int10_05()` 1334 `_int12()` 1334 `_int13_00()` 1334 `_int13_02()` 1334 `_int13_03()` 1334 `_int16_00()` 1334 `_int16_01()` 1334 `_int16_02()` 1334 `_in_16()` 1334 `_in_8()` 1334 `_out_16()` 1334 `_out_8()` 1334 `_ram_copy()` 1334 `_seg_d()` 1333 `_seg_i()` 1333 `_sp()` 1333 `_ss()` 1333

Il file '`kernel/ibm_i86.h`' e quelli contenuti nella directory '`kernel/ibm_i86/`', raccolgono il codice del kernel che è legato strettamente all'hardware; a questi file vanno anche aggiunti '`lib/sys/os16.h`' e la directory '`lib/sys/os16/`', della libreria, utilizzati anche dalle applicazioni, a vario titolo. In generale si può osservare la presenza di funzioni che si avvalgono direttamente di alcune interruzioni del BIOS (fondamentalmente per la gestione del video e per l'accesso ai dischi); funzioni che permettono di leggere il valore di alcuni registri; funzioni per leggere e scrivere la memoria, in posizioni arbitrarie; funzioni per facilitare la lettura e la scrittura nei dischi, anche a livello di byte.

Salvo poche eccezioni, le funzioni scritte in linguaggio assemblatore hanno nomi che iniziano con un trattino basso, ma a fianco di queste sono anche disponibili delle macroistruzioni, con nomi equivalenti, senza il trattino basso iniziale, per garantire che gli argomenti della chiamata abbiano il tipo corretto, restituendo un valore intero «normale», quando qualcosa deve essere restituito.

Libreria: «lib/sys/os16.h» e «lib/sys/os16/...»

Listato u0.12 e successivi.

Nel file '`lib/sys/os16.h`' e in quelli della directory '`lib/sys/os16/`' si raccolgono, tra le altre, delle funzioni di basso livello che possono essere utili per il kernel e per le applicazioni. Si tratta di ***_seg_i()*** e ***_seg_d()*** (ovvero le macroistruzioni ***seg_i()*** e ***seg_d()***), con cui si ottiene, rispettivamente, il numero del segmento codice (istruzioni) e il numero del segmento dati. Inoltre, per poter verificare gli altri registri di segmento e i registri di gestione della pila, si aggiungono le funzioni ***_cs()***, ***_ds()***, ***_ss()***, ***_es()***, ***_sp()*** e ***_bp()***; le quali, rispettivamente, consentono di leggere il valore dei registri ***CS***, ***DS***, ***SS***, ***ES***, ***SP*** e ***BP*** (le macroistruzioni equivalenti sono ***cs()***, ***ds()***, ***ss()***, ***es()***, ***sp()*** e ***bp()***).

Tabella u144.1. Funzioni e macroistruzioni legate strettamente all'hardware, dichiarate nel file di intestazione '`lib/sys/os16.h`'. Tali funzioni e macroistruzioni possono essere utilizzate sia dal kernel, sia dalle applicazioni.

Funzione o macroistruzione	Descrizione
`uint16_t _seg_i (void);` `unsigned int seg_i (void);`	Restituisce il numero del segmento codice (*instruction*). Listati u0.12 e i161.12.6.
`uint16_t _seg_d (void);` `unsigned int seg_d (void);`	Restituisce il numero del segmento dati. Listati u0.12 e i161.12.5.
`uint16_t _cs (void);` `unsigned int cs (void);`	Restituisce il valore del registro ***CS***. Listati u0.12 e i161.12.2.
`uint16_t _ds (void);` `unsigned int ds (void);`	Restituisce il valore del registro ***DS***. Listati u0.12 e i161.12.3.
`uint16_t _ss (void);` `unsigned int ss (void);`	Restituisce il valore del registro ***SS***. Listati u0.12 e i161.12.8.
`uint16_t _es (void);` `unsigned int es (void);`	Restituisce il valore del registro ***ES***. Listati u0.12 e i161.12.4.
`uint16_t _sp (void);` `unsigned int sp (void);`	Restituisce il valore del registro ***SP***. Il valore che si ottiene si riferisce allo stato del registro, prima di chiamare la funzione. Listati u0.12 e i161.12.7.
`uint16_t _bp (void);` `unsigned int bp (void);`	Restituisce il valore del registro ***BP***. Il valore che si ottiene si riferisce allo stato del registro, prima di chiamare la funzione. Listati u0.12 e i161.12.1.

Funzioni di basso livello dei file «kernel/ibm_i86/*»

« Listato u0.5 e successivi.

Le funzioni con nomi che iniziano per '**`_int`*nn*`…()`**', dove ***nn*** è un numero di due cifre, in base sedici, consentono l'accesso all'interruzione ***nn*** del BIOS dal codice in linguaggio C.

Le funzioni con nomi del tipo '**`_in_`*n*`()`**' e '**`_out_`*n*`()`**' consentono di leggere e di scrivere un valore di ***n*** bit in una certa porta.

La funzione ***cli()*** disabilita le interruzioni hardware, mentre ***sti()*** le riabilita. Queste due funzioni vengono usate pochissimo nel codice del kernel. A loro si aggiungono le funzioni ***irq_on()*** e ***irq_off()***, per abilitare o escludere selettivamente un tipo di interruzione hardware. Queste funzioni vengono usate in una sola occasione, quando si predispone la tabella IVT e poi si abilitano esclusivamente le interruzioni utili.

La funzione ***_ram_copy()*** si occupa di copiare una quantità stabilita di byte da una posizione della memoria a un'altra, entrambe indicate con segmento e scostamento (la funzione ***mem_copy()*** elencata in '`kernel/memory.h`' si avvale in pratica di questa).

Per agevolare l'uso di queste funzioni, senza costringere a convertire i valori numerici, sono disponibili diverse macroistruzioni con nomi equivalenti, ma privi del trattino basso iniziale.

Tabella u144.2. Funzioni e macroistruzioni di basso livello, dichiarate nel file di intestazione '`kernel/ibm_i86.h`' e descritte nei file della directory '`kernel/ibm_i860/`'. Le macroistruzioni hanno argomenti di tipo numerico non precisato, purché in grado di rappresentare il valore necessario.

Funzione o macroistruzione	Descrizione
`void _int10_00 (uint16_t` ***video_mode***`);` `void int10_00 (`***video_mode***`);`	Imposta la modalità video della console. Questa funzione viene usata solo da ***con_init()***, per inizializzare la console; la modalità video è stabilita dalla macro-variabile ***IBM_I86_VIDEO_MODE***, dichiarata nel file '`kernel/ibm_i86.h`'.

`void _int10_02 (uint16_t` ***page***`,`
`uint16_t` ***position***`);`
`void int10_02 (`***page***`,` ***position***`);`
	Colloca il cursore in una posizione determinata dello schermo, relativo a una certa pagina video. Questa funzione viene usata solo da ***con_putc()***.

`void _int10_05 (uint16_t` ***page***`);` `void int10_05 (`***page***`);`	Seleziona la pagina attiva del video. Questa funzione viene usata solo da ***con_init()*** e ***con_select()***.
`void _int12 (void);` `void int12 (void);`	Restituisce la quantità di memoria disponibile, in multipli di 1024 byte.
`void _int13_00 (uint16_t` ***drive***`);` `void int13_00 (`***drive***`);`	Azzera lo stato dell'unità a disco indicata, rappresentata da un numero secondo le convenzioni del BIOS. Viene usata solo dalle funzioni '**`dsk_…()`**' che si occupano dell'accesso alle unità a disco.

`uint16_t _int13_02 (uint16_t` ***drive***`,`
`uint16_t` ***sectors***`,`
`uint16_t` ***cylinder***`,`
`uint16_t` ***head***`,`
`uint16_t` ***sector***`,`
`void *`***buffer***`);`
`void int13_02 (`***drive***`,` ***sectors***`,`
cylinder`,` ***head***`,`
sector`,` ***buffer***`);`
	Legge dei settori da un'unità a disco. Questa funzione viene usata soltanto da ***dsk_read_sectors()***.

`uint16_t _int13_03 (uint16_t` ***drive***`,`
`uint16_t` ***sectors***`,`
`uint16_t` ***cylinder***`,`
`uint16_t` ***head***`,`
`uint16_t` ***sector***`,`
`void *`***buffer***`);`
`void int13_03 (`***drive***`,` ***sectors***`,`
cylinder`,` ***head***`,`
sector`,` ***buffer***`);`
	Scrive dei settori in un'unità a disco. Questa funzione viene usata solo da ***dsk_write_sectors()***.

`uint16_t _int16_00 (void);` `void int16_00 (void);`	Legge un carattere dalla tastiera, rimuovendolo dalla memoria tampone relativa. Viene usata solo in alcune funzioni di controllo della console, denominate '**`con_…()`**'.
`uint16_t _int16_01 (void);` `void int16_01 (void);`	Verifica se è disponibile un carattere dalla tastiera: se c'è ne restituisce il valore, ma senza rimuoverlo dalla memoria tampone relativa, altrimenti restituisce zero. Viene usata solo dalle funzioni di gestione della console, denominate '**`con_…()`**'.

Funzione o macroistruzione	Descrizione
`void _int16_02 (void);` `void int16_02 (void);`	Restituisce un valore con cui è possibile determinare quali funzioni speciali della tastiera risultano inserite (inserimento, fissamaiuscole, blocco numerico, ecc.). Al momento la funzione non viene usata.
`uint16_t _in_8 (uint16_t port);` `void in_8 (port);`	Legge un byte dalla porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.
`uint16_t _in_16 (uint16_t port);` `void in_16 (port);`	Legge un valore a 16 bit dalla porta di I/O indicata. Questa funzione viene usata solo da ***dev_mem()***.

`void _out_8 (uint16_t port,`
`uint16_t value);`
`void out_8 (port, value);`
	Scrive un byte nella porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.

`void _out_16 (uint16_t port,`
`uint16_t value);`
`void out_16 (port, value);`
	Scrive un valore a 16 bit nella porta indicata. Questa funzione viene usata solo da ***dev_mem()***.

`void cli (void);`	Azzera l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`CLI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void sti (void);`	Attiva l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`STI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void irq_on (unsigned int irq);`	Abilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.
`void irq_off (unsigned int irq);`	Disabilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.

`void _ram_copy (segment_t org_seg,`
`offset_t org_off,`
`segment_t dst_seg,`
`offset_t dst_off,`
`uint16_t size);`
`void ram_copy (org_seg,`
`org_off,`
`dst_seg,`
`dst_off,`
`size);`
	Copia una certa quantità di byte, da una posizione di memoria all'altra, specificando segmento e scostamento di origine e destinazione. Viene usata solo dalle funzioni '**`mem`**_...**`()`**'.

Gestione della console

«

Listato u0.5 e successivi.

La console offre solo funzionalità elementari, dove è possibile scrivere o leggere un carattere alla volta, sequenzialmente. Ci sono al massimo quattro console virtuali, selezionabili attraverso le combinazioni di tasti [*Ctrl q*], [*Ctrl r*], [*Ctrl s*] e [*Ctrl t*] (ma nella configurazione predefinita vengono attivate solo le prime due) e non è possibile controllare i colori o la posizione del testo che si va a esporre; in pratica si opera come su una telescrivente. Le funzioni di livello più basso, relative alla console hanno nomi che iniziano per '**`con`**_...**`()`**'.

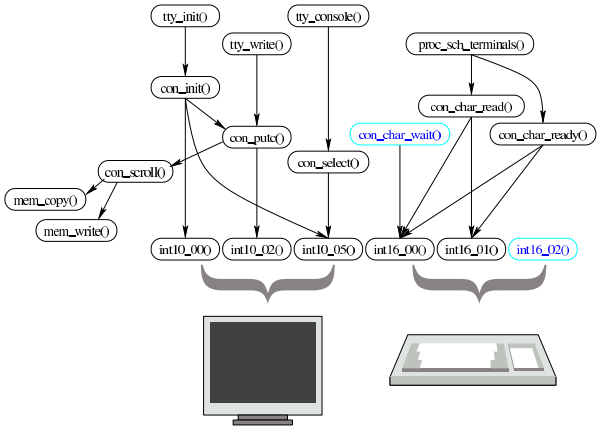
Nel codice del kernel si vede usata frequentemente la funzione ***k_printf()***, la quale va a utilizzare la funzione ***k_vprintf()***, dove poi, attraverso altri passaggi, si arriva a utilizzare la funzione ***con_putc()***.

Tabella u144.3. Funzioni per l'accesso alla console, dichiarate nel file di intestazione '`kernel/ibm_i86.h`' e descritte nei file contenuti nella directory '`kernel/ibm_i86/`'.

Funzione	Descrizione
`int con_char_read (void);`	Legge un carattere dalla console, se questo è disponibile, altrimenti restituisce il valore zero. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`int con_char_wait (void);`	Legge un carattere dalla console, ma se questo non è ancora disponibile, rimane in attesa, bloccando tutto il sistema operativo. Questa funzione non è utilizzata.
`int con_char_ready (void);`	Verifica se è disponibile un carattere dalla console: se è così, restituisce un valore diverso da zero, corrispondente al carattere in attesa di essere prelevato. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`void con_init (void);`	Inizializza la gestione della console. Questa funzione viene usata solo da ***tty_init()***.
`void con_select (int console);`	Seleziona la console desiderata, dove la prima si individua con lo zero. Questa funzione viene usata solo da ***tty_console()***.
`void con_putc (int console,` `int c);`	Visualizza il carattere indicato sullo schermo della console specificata, sulla posizione in cui si trova il cursore, facendolo avanzare di conseguenza e facendo scorrere il testo in alto, se necessario. Questa funzione viene usata solo da ***tty_write()***.
`void con_scroll (int console);`	Fa avanzare in alto il testo della console selezionata. Viene usata internamente, solo dalla funzione ***con_putc()***.

Nella figura successiva si vede l'interdipendenza tra le funzioni relative alla gestione di basso livello della console. In un altro capitolo si descrivono le funzioni '**`tty`**_...**`()`**', con le quali si gestiscono i terminali in forma più astratta. Nello schema successivo si può vedere che la funzione ***con_scroll()*** si avvale di funzioni per la gestione della memoria: infatti, lo scorrimento del testo dello schermo si ottiene intervenendo direttamente nella memoria utilizzata per la rappresentazione del testo sullo schermo.

Figura u144.4. Interdipendenza tra le funzioni relative alla gestione della console.

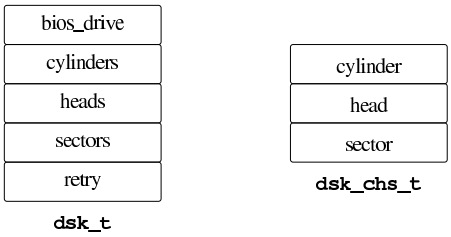


Gestione dei dischi

«

Listato u0.5 e successivi.

Nel file ‘`ibm_i86.h`’ vengono definiti due tipi derivati: ‘**`dsk_t`**’ per annotare le caratteristiche di un disco; ‘**`dsk_chs_t`**’ per annotare simultaneamente le coordinate di accesso a un disco, formate dal numero del cilindro, della testina e del settore.



Le funzioni con nomi del tipo ‘**`dsk_…()`**’ riguardano l’accesso ai dischi, a livello di settore o di byte, e utilizzano le informazioni annotate nell’array ***dsk_table[]***, composto da elementi di tipo ‘**`dsk_t`**’. In pratica, l’array ***dsk_table[]*** viene creato con ‘**`DSK_MAX`**’ elementi (pertanto solo quattro), uno per ogni disco che si intende gestire. Quando le funzioni ‘**`dsk_…()`**’ richiedono l’indicazione di un numero di unità (*drive*), si riferiscono all’indice dell’array ***dsk_table[]*** (al contrario, le funzioni ‘**`_int13_…()`**’ hanno come riferimento il codice usato dal BIOS).

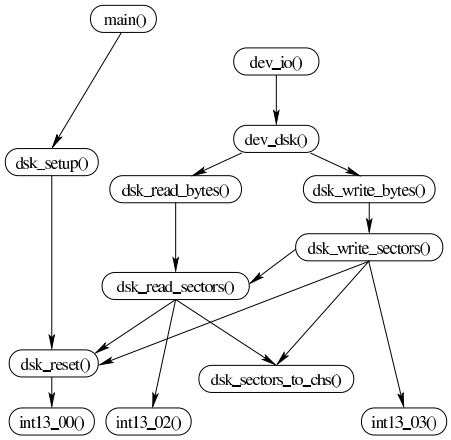
La funzione ***dsk_setup()*** compila l’array ***dsk_table[]*** con i dati relativi ai dischi che si utilizzano; la funzione ***dsk_reset()*** azzera la funzionalità di una certa unità; la funzione ***dsk_sector_to_chs()*** converte il numero assoluto di un settore nelle coordinate corrispondenti (cilindro, testina e settore).

Le funzioni ***dsk_read_sectors()*** e ***dsk_write_sectors()*** servono a leggere o scrivere una quantità stabilita di settori, usando come appoggio un’area di memoria individuata da un puntatore generico. Le funzioni ***dsk_read_bytes()*** e ***dsk_write_bytes()*** svolgono un compito equivalente, ma usando come riferimento il byte; in questo caso, restituiscono la quantità di byte letti o scritti rispettivamente.

Tabella u144.6. Funzioni per l’accesso ai dischi, dichiarate nel file di intestazione ‘`kernel/ibm_i86.h`’.

Funzione	Descrizione
`void dsk_setup (void);`	Predispone il contenuto dell’array ***dsk_table[]***. Questa funzione viene usata soltanto da ***main()***.
`int dsk_reset (int drive);`	Azzera lo stato dell’unità corrispondente a ***dsk_table[drive].bios_drive***. Viene usata solo internamente, dalle altre funzioni ‘**`dsk_…()`**’.
`void dsk_sector_to_chs (int drive, unsigned int sector, dsk_chs_t *chs);`	Modifica le coordinate della variabile strutturata a cui punta l’ultimo parametro, con le coordinate corrispondenti al numero di settore fornito. Viene usata solo internamente, dalle altre funzioni ‘**`dsk_…()`**’.
`int dsk_read_sectors (int drive, unsigned int start_sector, void *buffer, unsigned int n_sectors);`	Legge una sequenza di settori da un disco, mettendo i dati in memoria, a partire dalla posizione espressa da un puntatore generico. La funzione è ricorsiva, ma oltre che da se stessa, viene usata internamente da ***dsk_read_bytes()*** e da ***dsk_write_bytes()***.
`int dsk_write_sectors (int drive, unsigned int start_sector, void *buffer, unsigned int n_sectors);`	Scrive una sequenza di settori in un disco, traendo i dati da un puntatore a una certa posizione della memoria. La funzione è ricorsiva, ma oltre che da se stessa, viene usata solo internamente da ***dsk_write_bytes()***.
`size_t dsk_read_bytes (int drive, off_t offset, void *buffer, size_t count);`	Legge da una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.
`size_t dsk_write_bytes (int drive, off_t offset, void *buffer, size_t count);`	Scrive su una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.

Figura u144.7. Interdipendenza tra le funzioni relative all’accesso ai dischi.



File «kernel/memory.h» e «kernel/memory/...» 1341
Scansione della mappa di memoria 1343

`addr_t` 1341 `kernel/memory.c` 1341 `mb_alloc()` 1342 `mb_alloc_size()` 1342 `mb_free()` 1342 `mb_reference()` 1342 `memory.h` 1341 `memory_t` 1341 `MEM_BLOCK_SIZE` 1341 `mem_copy()` 1342 `MEM_MAX_BLOCKS` 1341 `mem_read()` 1342 `mem_write()` 1342 `offset_t` 1341 `segment_t` 1341

Dal punto di vista del kernel di os16, l'allocazione della memoria riguarda la collocazione dei processi elaborativi nella stessa. Disponendo di una quantità di memoria esigua, si utilizza una mappa di bit per indicare lo stato dei blocchi di memoria, dove un bit a uno indica un blocco di memoria occupato.

Nel file '`memory.h`' viene definita la dimensione di un blocco di memoria e, di conseguenza, la quantità massima che possa essere gestita. Attualmente i blocchi sono da 256 byte, pertanto, sapendo che la memoria può arrivare solo fino a 640 Kibyte, si gestiscono al massimo 2560 blocchi.

Per la scansione della mappa si utilizzano interi da 16 bit, pertanto tutta la mappa si riduce a 40 di questi interi, ovvero 80 byte. Nell'ambito di ogni intero da 16 bit, il bit più significativo rappresenta il primo blocco di memoria di sua competenza. Per esempio, per indicare che si stanno utilizzando i primi 1280 byte, pari ai primi cinque blocchi di memoria, si rappresenta la mappa della memoria come «F80000000...».

Il fatto che la mappa della memoria vada scandito a ranghi di 16 bit va tenuto in considerazione, perché se invece si andasse con ranghi differenti, si incapperebbe nel problema dell'inversione dei byte.

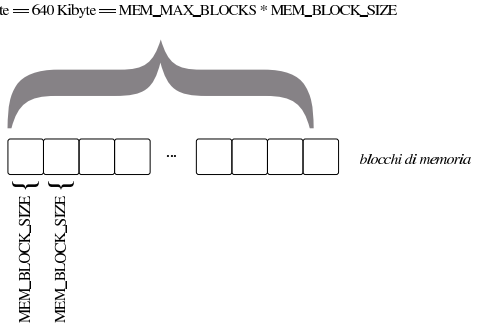
Quando possibile, si fa riferimento a indirizzi di memoria efficaci, nel senso che, con un solo valore, si rappresentano le posizioni da 00000_{16} a $FFFFF_{16}$. Per questo viene predisposto il tipo derivato '**`addr_t`**' nel file '`memory.h`'.

File «kernel/memory.h» e «kernel/memory/...»

Listato u0.8 e successivi.

Il file '`kernel/memory.h`', oltre ai prototipi delle funzioni usate per la gestione della memoria, definisce la dimensione del blocco minimo di memoria e la quantità massima di questi, rispettivamente con le macro-variabili ***MEM_BLOCK_SIZE*** e ***MEM_MAX_BLOCKS***; inoltre predispone i tipi derivati '**`memory_t`**', '**`segment_t`**', '**`offset_t`**' e '**`addr_t`**', corrispondenti, rispettivamente, a una variabile strutturata che consente di rappresentare un'area di memoria in modo relativamente comodo (indirizzo efficace, segmento e dimensione), un indirizzo di segmento, uno scostamento dall'inizio di un segmento e un indirizzo efficace.

Figura u145.1. Mappa della memoria in blocchi: la dimensione minima di un'area di memoria è di '**`MEM_BLOCK_SIZE`**' byte.



Nei file della directory '`kernel/memory/`' viene dichiarata la mappa della memoria, corrispondente a un array di interi a 16 bit, denominato ***mb_table[]***. L'array è pubblico, tuttavia è disponibile anche «

una funzione che ne restituisce il puntatore: ***mb_reference()***. Tale funzione sarebbe perfettamente inutile, ma rimane per uniformità rispetto alla gestione delle altre tabelle.

Nelle funzioni che riguardano l'allocazione della memoria, quando si indica la dimensione di questa, spesso si considera il valore zero equivalente a 10000_{16}, ovvero la dimensione massima di un segmento secondo l'architettura.

Tabella u145.2. Funzioni per la gestione della mappa della memoria, dichiarate nel file di intestazione '`kernel/memory.h`' e realizzate nella directory '`kernel/memory/`'.

Funzione	Descrizione
`uint16_t *mb_reference (void);`	Restituisce il puntatore alla tabella dei blocchi di memoria, per uniformare l'accesso alla tabella dalle funzioni che non fanno parte del gruppo contenuto nella directory '`kernel/memory/`'.
`ssize_t mb_alloc (addr_t address, size_t size);`	Alloca la memoria a partire dall'indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). L'allocazione ha termine anticipatamente se si incontra un blocco già utilizzato. La funzione restituisce la dimensione allocata effettivamente.
`ssize_t mb_free (addr_t address, size_t size);`	Libera la memoria a partire dall'indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). Lo spazio viene liberato in ogni caso, anche se risulta già libero; tuttavia viene prodotto un avvertimento a video se si verifica tale ipotesi.
`int mb_alloc_size (size_t size, memory_t *allocated);`	Cerca e alloca un'area di memoria della dimensione richiesta, modificando la variabile strutturata di cui viene fornito il puntatore come secondo parametro. In pratica, l'indirizzo e l'estensione della memoria allocata effettivamente si trovano nella variabile strutturata in questione, mentre la funzione restituisce zero (se va tutto bene) o `-1` se non è disponibile la memoria libera richiesta.

Tabella u145.3. Funzioni per le operazioni di lettura e scrittura in memoria, dichiarate nel file di intestazione '`kernel/memory.h`' e realizzate nella directory '`kernel/memory/`'.

Funzione	Descrizione
`void mem_copy (addr_t orig, addr_t dest, size_t size);`	Copia la quantità richiesta di byte, dall'indirizzo di origine a quello di destinazione, espressi in modo efficace.

Funzione	Descrizione
`size_t mem_read (addr_t start, void *buffer, size_t size);`	Legge dalla memoria, a partire dall'indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene letto va poi copiato nella memoria tampone corrispondente al puntatore generico indicato come secondo parametro.
`size_t mem_write (addr_t start, void *buffer, size_t size);`	Scrive, in memoria, a partire dall'indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene scritto proviene dalla memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

Scansione della mappa di memoria

Listato u0.8 e successivi.

La mappa della memoria si rappresenta (a sua volta in memoria), con un array di interi a 16 bit, dove ogni bit individua un blocco di memoria. Pertanto, l'array si compone di una quantità di elementi pari al valore di '**`MEM_MAX_BLOCKS`**' diviso 16.

Il primo elemento di questo array, ovvero ***mb_table[0]***, individua i primi 16 blocchi di memoria, dove il bit più significativo si riferisce precisamente al primo blocco. Per esempio, se ***mb_table[0]*** contiene il valore $F800_{16}$, ovvero 1111100000000000_2, significa che i primi cinque blocchi di memoria sono occupati, mentre i blocchi dal sesto al sedicesimo sono liberi.

Dal momento che i calcoli per individuare i blocchi di memoria e per intervenire nella mappa relativa, possono creare confusione, queste operazioni sono raccolte in funzioni statiche separate, anche se sono utili esclusivamente all'interno del file in cui si trovano. Tali funzioni statiche hanno una sintassi comune:

```
int mb_block_set1   (int block)
int mb_block_set0   (int block)
int mb_block_status (int block)
```

Le funzioni ***mb_block_set1()*** e ***mb_block_set0()*** servono rispettivamente a impegnare o liberare un certo blocco di memoria, individuato dal valore dell'argomento. La funzione ***mb_block_status()*** restituisce uno nel caso il blocco indicato risulti allocato, oppure zero in caso contrario.

Queste tre funzioni usano un metodo comune per scandire la mappa della memoria: il valore che rappresenta il blocco a cui si vuole fare riferimento, viene diviso per 16, ovvero il rango degli elementi dell'array che rappresenta la mappa della memoria. Il risultato intero della divisione serve per trovare quale elemento dell'array considerare, mentre il resto della divisione serve per determinare quale bit dell'elemento trovato rappresenta il blocco desiderato. Trovato ciò, si deve costruire una maschera, nella quale si mette a uno il bit che rappresenta il blocco; per farlo, si pone inizialmente a uno il bit più significativo della maschera, quindi lo si fa scorrere verso destra di un valore pari al resto della divisione.

Per esempio, volendo individuare il terzo blocco di memoria, pari al numero 2 (il primo blocco corrisponderebbe allo zero), si avrebbe che questo è descritto dal primo elemento dell'array (in quanto 2/16 dà zero, come risultato intero), mentre la maschera necessaria a trovare il bit corrispondente è 0010000000000000_2, la quale si ottiene spostando per due volte verso destra il bit più significativo (due volte, pari al resto della divisione).

Una volta determinata la maschera, per segnare come occupato un blocco di memoria, basta utilizzare l'operatore OR binario:

```
mb_table[i] = mb_table[i] | mask;
```

Se invece si vuole liberare un blocco di memoria, si utilizza un AND binario, invertendo però il contenuto della maschera:

```
mb_table[i] = mb_table[i] & ~mask;
```

Va osservato che la rappresentazione dei blocchi nella mappa è invertita rispetto ad altri sistemi operativi, in quanto non sarebbe tanto logico il fatto che il bit più significativo si riferisca invece alla parte più bassa del proprio insieme di blocchi di memoria. La scelta è dovuta al fatto che, volendo rappresentare la mappa numericamente, la lettura di questa sarebbe più vicina a quella che è la percezione umana del problema.

`tty.h` 1345 `tty_console()` 1345 `tty_init()` 1345 `tty_read()` 1345 `tty_reference()` 1345 `tty_write()` 1345

Listato u0.10 e successivi.

os16 offre esclusivamente un utilizzo operativo tramite console. Alcune funzioni con prefisso '**`con`**`_...()`', dichiarate nel file '`kernel/ibm_i86.h`', si occupano di tale gestione, ma per distinguere tra terminali virtuali (o console virtuali), associati a processi differenti, si rende necessario un livello ulteriore di astrazione, costituito dal codice contenuto nel file '`kernel/tty.h`' e in quelli della directory '`kernel/tty/`'.

I terminali virtuali gestibili sono rappresentati da un array di variabili strutturate, ognuna delle quali contiene tutte le informazioni del contesto operativo di un certo terminale. L'array in questione è ***tty_table[]*** (a cui però si accede tramite una funzione che ne restituisce il puntatore) e vi si annotano, per ogni terminale attivo, il numero del dispositivo corrispondente, il numero del gruppo di processi a cui si associa, l'ultimo codice digitato (e non ancora letto), lo stato di funzionamento (se sono stati persi dei dati o meno).

Va osservata la differenza sostanziale che c'è tra le operazioni di scrittura e quelle di lettura. Infatti, la scrittura sul terminale implica la chiamata della funzione ***con_putc()***, del file '`kernel/ibm_i86.h`'; al contrario, la lettura avviene in forma passiva, limitandosi ad acquisire il valore già disponibile nella variabile strutturata che rappresenta il terminale virtuale. Come può essere verificato successivamente, è compito del sistema di gestione delle interruzioni la fornitura del valore digitato al terminale virtuale competente, tramite l'appoggio della variabile strutturata che lo rappresenta.

Come per tutte le tabelle di os16 che non fanno parte di uno standard, anche quella che contiene le informazioni dei terminali virtuali è accessibile preferibilmente con l'ausilio di una funzione che ne restituisce il puntatore. In questo caso, la funzione ***tty_reference()*** consente di ottenere il puntatore all'elemento corrispondente della tabella dei terminali, fornendo come argomento il numero del dispositivo cercato.

Tabella u146.1. Funzioni per la gestione dei terminali, dichiarate nel file di intestazione '`kernel/tty.h`'.

Funzione	Descrizione
`void tty_init (void);`	Inizializza la gestione dei terminali. Viene usata una volta sola nella funzione ***main()*** del kernel.
`tty_t *tty_reference (dev_t` ***device***`);`	Restituisce il puntatore a un elemento della tabella dei terminali. Se come numero di dispositivo si indica lo zero, si ottiene il riferimento a tutta la tabella; se non viene trovato il numero di dispositivo cercato, si ottiene il puntatore nullo.

Funzione	Descrizione
`dev_t tty_console (dev_t` ***device*** `);`	Seleziona la console indicata attraverso il numero di dispositivo che costituisce l'unico parametro. Se viene dato un valore a zero, si ottiene solo di conoscere qual è la console attiva. La console selezionata viene anche memorizzata in una variabile statica, per le chiamate successive della funzione. Se viene indicato un numero di dispositivo non valido, si seleziona implicitamente la prima console.
`int tty_read (dev_t` ***device*** `);`	Legge un carattere dal terminale specificato attraverso il numero di dispositivo. Per la precisione, il carattere viene tratto dal campo relativo contenuto nella tabella dei terminali. Il carattere viene restituito dalla funzione come valore intero comune; se si ottiene zero significa che non è disponibile alcun carattere.
`void tty_write (dev_t` ***device***`, int` ***c***`);`	Scrive sullo schermo del terminale rappresentato dal numero di dispositivo, il carattere fornito come secondo parametro.

Dispositivi

File «lib/sys/os16.h» e directory «lib/sys/os16/» 1347
File «kernel/devices.h» e «kernel/devices/...» 1347
Numero primario e numero secondario 1348
Dispositivi previsti 1349

`devices.h` 1347 `devices/dev_tty.c` 1347 `DEV_CONSOLE` 1349 `DEV_CONSOLE`***n*** 1349 `dev_dsk.c` 1347 `DEV_DSK`***n*** 1349 `dev_io()` 1347 `dev_io.c` 1347 `dev_kmem.c` 1347 `DEV_KMEM_FILE` 1349 `DEV_KMEM_INODE` 1349 `DEV_KMEM_MMP` 1349 `DEV_KMEM_PS` 1349 `DEV_KMEM_SB` 1349 `DEV_MEM` 1349 `DEV_NULL` 1349 `DEV_PORT` 1349 `DEV_TTY` 1349 `DEV_ZERO` 1349 `os16.h` 1347

La gestione dei dispositivi fisici, da parte di os16, è molto limitata, in quanto ci si avvale prevalentemente delle funzionalità già offerte dal BIOS. In ogni caso, tutte le operazioni di lettura e scrittura di dispositivi, passano attraverso la gestione comune della funzione ***dev_io()***.

File «lib/sys/os16.h» e directory «lib/sys/os16/»

Listato u0.12 e altri.

Una parte delle definizioni che riguardano la gestione dei dispositivi, necessaria al kernel, è disponibile anche alle applicazioni attraverso il file '`lib/sys/os16.h`'. Al suo interno, tra le altre cose, è definito un insieme di macro-variabili, con prefisso ***DEV_***..., con cui si distinguono i numeri attribuiti ai dispositivi. Per esempio, ***DEV_DSK_MAJOR*** corrisponde al numero primario (*major*) per tutte le unità di memorizzazione, mentre ***DEV_DSK1*** corrisponde al numero primario e secondario (*major* e *minor*), in un valore unico, della seconda unità a disco.

File «kernel/devices.h» e «kernel/devices/...»

Listati u0.2 e altri.

Il file '`kernel/devices.h`' incorpora il file '`lib/sys/os16/os16.h`', per acquisire le funzionalità legate alla gestione dei dispositivi che sono disponibili anche agli applicativi. Successivamente dichiara la funzione ***dev_io()***, la quale sintetizza tutta la gestione dei dispositivi. Questa funzione utilizza il parametro ***rw***, per specificare l'azione da svolgere (lettura o scrittura). Per questo parametro vanno usate le macro-variabili ***DEV_READ*** e ***DEV_WRITE***, così da non dover ricordare quale valore numerico corrisponde alla lettura e quale alla scrittura.

```
ssize_t dev_io (pid_t pid, dev_t device, int rw, off_t offset,
                void *buffer, size_t size, int *eof);
```

Sono comunque descritte anche altre funzioni, ma utilizzate esclusivamente da ***dev_io()***.

La funzione ***dev_io()*** si limita a estrapolare il numero primario dal numero del dispositivo complessivo, quindi lo confronta con i vari tipi gestibili. A seconda del numero primario seleziona una funzione appropriata per la gestione di quel tipo di dispositivo, passando praticamente gli stessi argomenti già ricevuti.

Va osservato il caso particolare dei dispositivi '**`DEV_KMEM`**_...'. In un sistema operativo Unix comune, attraverso ciò che fa capo al file di dispositivo '`/dev/kmem`', si ha la possibilità di accedere all'immagine in memoria del kernel, lasciando a un programma con privilegi adeguati la facoltà di interpretare i simboli che consentono di individuare i dati esistenti. Nel caso di os16, non ci sono simboli nel risultato della compilazione, quindi non è possibile ricostruire la collocazione dei dati. Per questa ragione, le informazioni che devono essere pubblicate, vengono controllate attraverso un dispositivo specifico. Quindi, il dispositivo '**`DEV_KMEM_PS`**' consente di leggere la tabella dei processi, '**`DEV_KMEM_MMAP`**' consente di leggere la mappa della memoria, e così vale anche per altre tabelle.

Per quanto riguarda la gestione dei terminali, attraverso la funzione ***dev_tty()***, quando un processo vuole leggere dal terminale, ma non risulta disponibile un carattere, questo viene messo in pausa, in attesa di un evento legato ai terminali.

os16, non disponendo di un sistema di trattenimento dei dati in memoria (*cache*), esegue le operazioni di lettura e scrittura dei dispositivi in modo immediato. Per questo motivo, la distinzione tra file di dispositivo a blocchi e a caratteri, rimane puramente estetica, ovvero priva di un'utilità concreta.

Figura u147.1. Interdipendenza tra la funzione ***dev_io()*** e le altre. I collegamenti con le funzioni ***major()*** e ***minor()*** sono omesse.

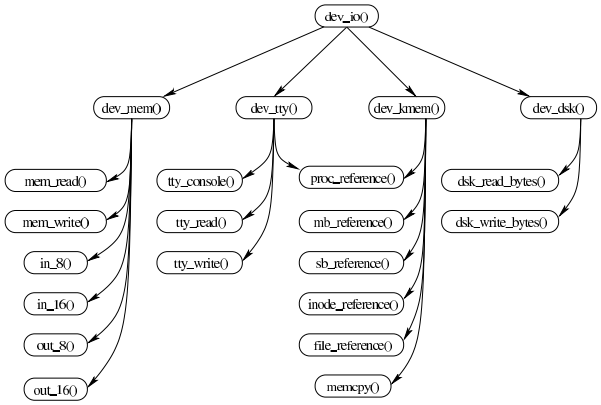
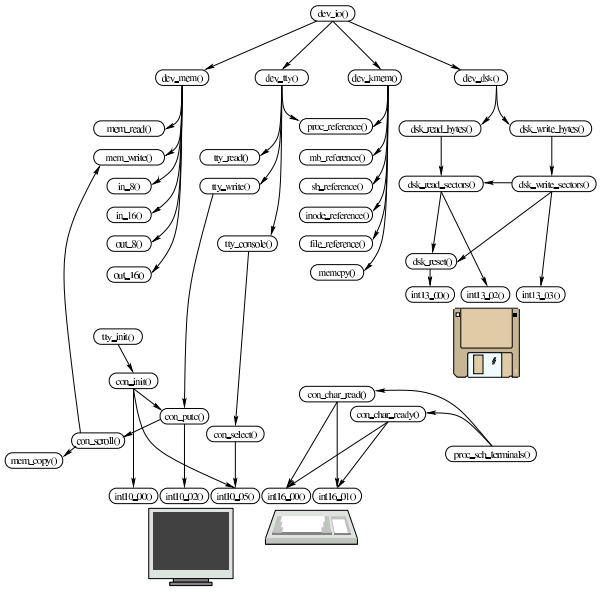


Figura u147.2. Schema più ampio delle dipendenze che hanno origine dalla funzione ***dev_io()***.



Numero primario e numero secondario

I dispositivi, secondo la tradizione dei sistemi Unix, sono rappresentati dal punto di vista logico attraverso un numero intero, senza segno, a 16 bit. Tuttavia, per organizzare questa numerazione in modo ordinato, tale numero viene diviso in due parti: la prima parte, nota come *major*, ovvero «numero primario», si utilizza per individuare il tipo di dispositivo; la seconda, nota come *minor*, ovvero «numero secondario», si utilizza per individuare precisamente il dispositivo, nell'ambito del tipo a cui appartiene.

In pratica, il numero complessivo a 16 bit si divide in due, dove gli 8 bit più significativi individuano il numero primario, mentre quelli meno significativi danno il numero secondario. L'esempio seguente si riferisce al dispositivo che genera il valore zero, il quale appartiene al gruppo dei dispositivi relativi alla memoria:

`DEV_MEM_MAJOR`	01_{16}
`DEV_ZERO`	0104_{16}

In questo caso, il valore che rappresenta complessivamente il dispositivo è 0104_{16} (pari a 260_{10}), ma si compone di numero primario 01_{16} e di numero secondario 04_{16} (che coincidono nella rappresentazione in base dieci). Per estrarre il numero primario si deve dividere il numero complessivo per 256 (0100_{16}), trattenendo soltanto il risultato intero; per filtrare il numero secondario si può fare la stessa divisione, ma trattenendo soltanto il resto della stessa. Al contrario, per produrre il numero del dispositivo, partendo dai numeri primario e secondario separati, occorre moltiplicare il numero primario per 256, sommando poi il risultato al numero secondario.

Dispositivi previsti

L'astrazione della gestione dei dispositivi, consente di trattare tutti i componenti che hanno a che fare con ingresso e uscita di dati, in modo sostanzialmente omogeneo; tuttavia, le caratteristiche effettive di tali componenti può comportare delle limitazioni o delle peculiarità. Ci sono alcune questioni fondamentali da considerare: un tipo di dispositivo potrebbe consentire l'accesso in un solo verso (lettura o scrittura); l'accesso al dispositivo potrebbe essere ammesso solo in modo sequenziale, rendendo inutile l'indicazione di un indirizzo; la dimensione dell'informazione da trasferire potrebbe assumere un significato differente rispetto a quello comune.

Tabella u147.4. Classificazione dei dispositivi di os16.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_MEM`	r/w	diretto	Permette l'accesso alla memoria, in modo indiscriminato, perché os16 non offre alcun tipo di protezione al riguardo.
`DEV_NULL`	r/w	nessuno	Consente la lettura e la scrittura, ma non si legge e non si scrive alcunché.
`DEV_PORT`	r/w	sequenziale	Consente di leggere e scrivere da o verso una porta di I/O, individuata attraverso l'indirizzo di accesso (l'indirizzo, o meglio lo scostamento, viene trattato come la porta a cui si vuole accedere). Tuttavia, la dimensione dell'informazione da trasferire è valida solo se si tratta di uno o di due byte: per la dimensione di un byte si usano le funzioni ***in_8()*** e ***out_8()***; per due byte si usano le funzioni ***in_16()*** e ***out_16()***. Per dimensioni differenti la lettura o la scrittura non ha effetto.
`DEV_ZERO`	r	sequenziale	Consente solo la lettura di valori a zero (zero inteso in senso binario).
`DEV_TTY`	r/w	sequenziale	Rappresenta il terminale virtuale del processo attivo.
`DEV_DSK`***n***	r/w	diretto	Rappresenta l'unità a dischi ***n***. os16 non gestisce le partizioni.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_PS`	r	diretto	Rappresenta la tabella contenente le informazioni sui processi. L'indirizzo di accesso indica il numero del processo di partenza; la dimensione da leggere dovrebbe essere abbastanza grande da contenere un processo, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_MMP`	r	sequenziale	Rappresenta la mappa della memoria, alla quale si può accedere solo dal suo principio. In pratica, l'indirizzo di accesso viene ignorato, mentre conta solo la quantità di byte richiesta.
`DEV_KMEM_SB`	r	diretto	Rappresenta la tabella dei super blocchi (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il super blocco; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un super blocco, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_INODE`	r	diretto	Rappresenta la tabella degli inode (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare l'inode; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un inode, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_FILE`	r	diretto	Rappresenta la tabella dei file (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il file; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di un file, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_CONSOLE`	r/w	sequenziale	Legge o scrive relativamente alla console attiva la quantità di byte richiesta, ignorando l'indirizzo di accesso.
`DEV_CONSOLE`***n***	r/w	sequenziale	Legge o scrive relativamente alla console ***n*** la quantità di byte richiesta, ignorando l'indirizzo di accesso.

File «kernel/fs/sb_...» 1351
File «kernel/fs/zone_...» 1353
File «kernel/fs/inode_...» 1354
Fasi dell'innesto di un file system 1357
File «kernel/fs/file_...» 1358
Descrittori di file 1359
File «kernel/fs/path_...» 1360
File «kernel/fs/fd_...» 1362

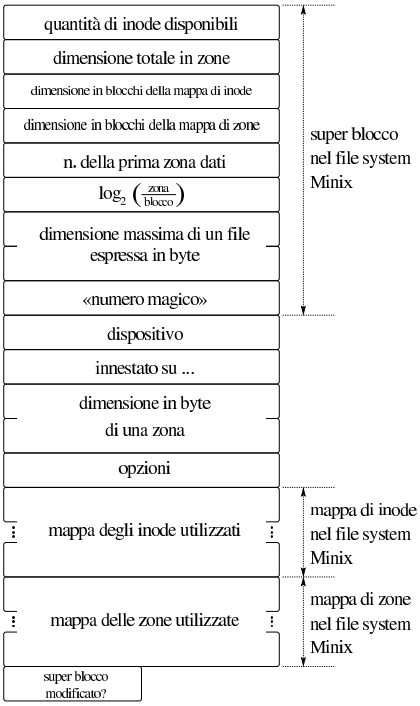
`fd_chmod()` 1362 `fd_chown()` 1362 `fd_close()` 1362 `fd_dup()` 1362 `fd_dup2()` 1362 `fd_fcntl()` 1362 `fd_lseek()` 1362 `fd_open()` 1362 `fd_read()` 1362 `fd_reference()` 1362 `fd_stat()` 1362 `fd_write()` 1362 `file_reference()` 1359 `file_stdio_dev_make()` 1359 `file_t` 1358 `fs.h` 1351 `inode_alloc()` 1355 `inode_check()` 1355 `inode_dir_empty()` 1355 `inode_file_read()` 1355 `inode_file_write()` 1355 `inode_fzones_read()` 1355 `inode_get()` 1355 `inode_put()` 1355 `inode_reference()` 1355 `inode_save()` 1355 `inode_t` 1354 `inode_truncate()` 1355 `inode_zone()` 1355 `path_chdir()` 1361 `path_chmod()` 1361 `path_chown()` 1361 `path_device()` 1361 `path_fix()` 1360 `path_full()` 1360 `path_inode()` 1361 `path_inode_link()` 1361 `path_link()` 1361 `path_mkdir()` 1361 `path_mknod()` 1361 `path_mount()` 1361 `path_stat()` 1361 `path_umount()` 1361 `path_unlink()` 1361 `sb_inode_status()` 1352 `sb_mount()` 1352 `sb_reference()` 1352 `sb_save()` 1352 `sb_t` 1351 `sb_zone_status()` 1352 `zone_alloc()` 1354 `zone_free()` 1354 `zone_read()` 1354 `zone_write()` 1354

La gestione del file system è suddivisa in diversi file contenuti nella directory '`kernel/fs/`', facenti capo al file di intestazione '`kernel/fs.h`'.

File «kernel/fs/sb_...»

I file '`kernel/fs/sb_...`' descrivono le funzioni per la gestione dei super blocchi, distinguibili perché iniziano tutte con il prefisso '**`sb_`**'. Tra questi file si dichiara l'array ***sb_table[]***, il quale rappresenta una tabella le cui righe sono rappresentate da elementi di tipo '**`sb_t`**' (il tipo '**`sb_t`**' è definito nel file '`kernel/fs.h`'). Per uniformare l'accesso alla tabella, la funzione ***sb_reference()*** permette di ottenere il puntatore a un elemento dell'array ***sb_table[]***, specificando il numero del dispositivo cercato.

Figura u148.1. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.



Listato u148.2. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.

```
typedef struct sb        sb_t;

struct sb {
  uint16_t  inodes;
  uint16_t  zones;
  uint16_t  map_inode_blocks;
  uint16_t  map_zone_blocks;
  uint16_t  first_data_zone;
  uint16_t  log2_size_zone;
  uint32_t  max_file_size;
  uint16_t  magic_number;
  //-------------------------
  dev_t     device;
  inode_t  *inode_mounted_on;
  blksize_t blksize;
  int       options;
  uint16_t  map_inode[SB_MAP_INODE_SIZE];
  uint16_t  map_zone[SB_MAP_ZONE_SIZE];
  char      changed;
};
```

Il super blocco rappresentato dal tipo '**sb_t**' include anche le mappe delle zone e degli inode impegnati. Queste mappe hanno una dimensione fissa in memoria, mentre nel file system reale possono essere di dimensione minore. La tabella di super blocchi, contiene le informazioni dei dispositivi di memorizzazione innestati nel sistema. L'innesto si concretizza nel riferimento a un inode, contenuto nella tabella degli inode (descritta in un altro capitolo), il quale rappresenta la directory di un'altra unità, su cui tale innesto è avvenuto. Naturalmente, l'innesto del file system principale rappresenta un caso particolare.

Tabella u148.3. Funzioni per la gestione dei dispositivi di memorizzazione di massa, a livello di super blocco, definite nei file '`kernel/fs/sb_...`'.

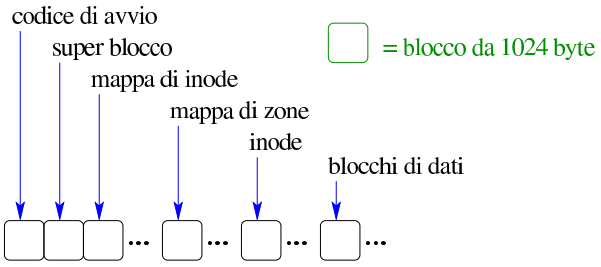
Funzione	Descrizione
`sb_t *sb_reference (dev_t` ***device***`);`	Restituisce il riferimento a un elemento della tabella dei super blocchi, in base al numero del dispositivo di memorizzazione. Se il dispositivo cercato non risulta già innestato, si ottiene il puntatore nullo; se si chiede il dispositivo zero, si ottiene il puntatore al primo elemento della tabella.

`sb_t *sb_mount (dev_t` ***device***`,`
`inode_t **`***inode_mnt***`,`
`int` ***options***`);`
	Innesta il dispositivo rappresentato numericamente dal primo parametro, sulla directory corrispondente all'inode a cui punta il secondo parametro, con le opzioni del terzo parametro. Quando si tratta del primo innesto del file system principale, la directory è quella dello stesso file system, pertanto, in tal caso, ****inode_mnt*** è inizialmente un puntatore nullo e deve essere modificato dalla funzione stessa.

`int sb_save (sb_t *`***sb***`);`	Salva il super blocco nella sua unità di memorizzazione, se questo risulta modificato. In questo caso, il super blocco include anche le mappe degli inode e delle zone.
`int sb_zone_status (sb_t *`***sb***`,` `zno_t` ***zone***`);`	Restituisce uno se la zona rappresentata dal secondo parametro è impegnata nel super blocco a cui si riferisce il primo parametro; diversamente restituisce zero.
`int sb_inode_status (sb_t *`***sb***`,` `ino_t` ***ino***`);`	Restituisce uno se l'inode rappresentato dal secondo parametro è impegnato nel super blocco a cui si riferisce il primo parametro; diversamente restituisce zero.

File «kernel/fs/zone_...»

Nel file system Minix 1, si distinguono i concetti di blocco e zona di dati, con il vincolo che la zona ha una dimensione multipla del blocco. Il contenuto del file system, dopo tutte le informazioni amministrative, è organizzato in zone; in altri termini, i blocchi di dati si raggiungono in qualità di zone.



La zona rimane comunque un tipo di blocco, potenzialmente più grande (ma sempre multiplo) del blocco vero e proprio, che si numera a partire dall'inizio dello spazio disponibile, con la differenza che

è utile solo per raggiungere i blocchi di dati. Nel super blocco del file system si trova l'informazione del numero della prima zona che contiene dati, in modo da non dover ricalcolare questa informazione ogni volta.

I file '`kernel/fs/zone_...`' descrivono le funzioni per la gestione del file system a zone.

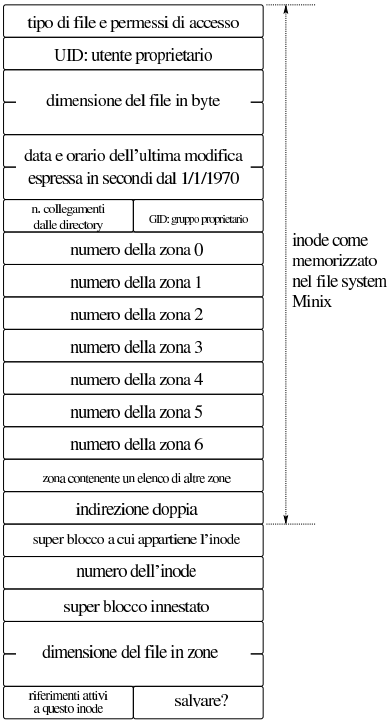
Tabella u148.5. Funzioni per la gestione delle zone, definite nei file '`kernel/fs/zone_...`'.

Funzione	Descrizione
`zno_t zone_alloc (sb_t *sb);`	Alloca una zona, restituendo il numero della stessa. In pratica, cerca la prima zona libera nel file system a cui si riferisce il super blocco ***sb** e la segna come impegnata, restituendone il numero.
`int zone_free (sb_t *sb, zno_t zone);`	Libera una zona, impegnata precedentemente.
`int zone_read (sb_t *sb, zno_t zone, void *buffer);`	Legge il contenuto di una zona, memorizzandolo a partire dalla posizione di memoria rappresentato da ***buffer***.
`int zone_write (sb_t *sb, zno_t zone, void *buffer);`	Sovrascrive una zona, utilizzando il contenuto della memoria a partire dalla posizione rappresentata da ***buffer***.

File «kernel/fs/inode_...»

I file '`kernel/fs/inode_...`' descrivono le funzioni per la gestione dei file, in forma di inode. In uno di questi file viene dichiarata la tabella degli inode in uso nel sistema, rappresentata dall'array ***inode_table[]*** e per individuare un certo elemento dell'array si usa preferibilmente la funzione ***inode_reference()***. Gli elementi della tabella degli inode sono di tipo '**`inode_t`**' (definito nel file '`kernel/fs.h`'); una voce della tabella rappresenta un inode utilizzato se il campo dei riferimenti (***references***) ha un valore maggiore di zero.

Figura u148.6. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.



Listato u148.7. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.

```
<verbatimpre width="60">
<![CDATA[
typedef struct inode     inode_t;

struct inode {
    mode_t        mode;
    uid_t         uid;
    ssize_t       size;
    time_t        time;
    uint8_t       gid;
    uint8_t       links;
    zno_t         direct[7];
    zno_t         indirect1;
    zno_t         indirect2;
    //------------------------
    sb_t         *sb;
    ino_t         ino;
    sb_t         *sb_attached;
    blkcnt_t      blkcnt;
    unsigned char references;
    char          changed;
};
```

Figura u148.8. Collegamento tra la tabella degli inode e quella dei super blocchi.

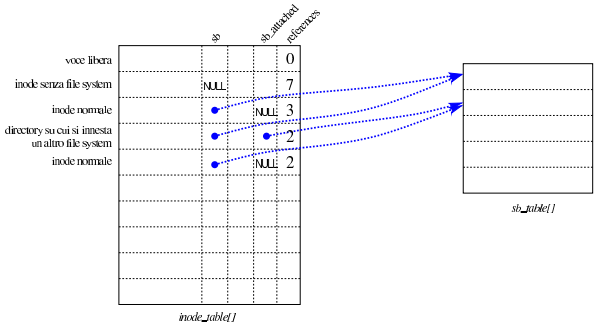


Tabella u148.9. Funzioni per la gestione dei file in forma di inode, definite nei file '`kernel/fs/inode_...`'.

Funzione	Descrizione
`inode_t *` `inode_reference (dev_t device, ino_t ino);`	Restituisce il puntatore a un inode, rappresentato in pratica da un elemento dell'array ***inode_table[]***, corrispondente a quello con il numero di dispositivo e di inode indicati come argomenti. Se entrambi gli argomenti sono a zero, si ottiene il puntatore al primo elemento; se entrambi i valori sono pari a `-1`, si ottiene il puntatore al primo elemento libero; se viene indicato il dispositivo zero e l'inode numero uno, si ottiene il puntatore all'elemento corrispondente alla directory radice del file system principale.

Funzione	Descrizione

`inode_t *`
`inode_alloc (dev_t` ***device***`,`
`mode_t` ***mode***`,`
`uid_t` ***uid***`);`
	La funzione ***inode_alloc()*** cerca un inode libero nel file system del dispositivo indicato, quindi lo alloca (lo segna come utilizzato) e lo modifica aggiornando il tipo e la modalità dei permessi, oltre al proprietario del file. Se la funzione riesce nel suo intento, restituisce il puntatore all'inode in memoria, il quale rimane così aperto e disponibile per ulteriori elaborazioni.

`inode_t *`
`inode_get (dev_t` ***device***`,`
`ino_t` ***ino***`);`
	Restituisce il puntatore all'inode rappresentato dal numero di dispositivo e di inode, indicati come argomenti. Se l'inode è già presente nella tabella degli inode, la cosa si risolve nell'incremento di una unità del numero dei riferimenti di tale inode; se invece l'inode non è ancora presente, questo viene caricato dal suo file system nella tabella e gli viene attribuito inizialmente un riferimento attivo.

`int inode_put (inode_t *`***inode***`);`	Rilascia un inode che non serve più. Ciò comporta la riduzione del contatore dei riferimenti nella tabella degli inode, tenendo conto che se tale valore raggiunge lo zero, si provvede anche al suo salvataggio nel file system (ammesso che l'inode della tabella risulti modificato, rispetto alla versione presente nel file system). La funzione restituisce zero in caso di successo, oppure `-1` in caso contrario.
`int inode_save (inode_t *`***inode***`);`	Salva l'inode nel file system, se questo risulta modificato.

`blkcnt_t`
`inode_fzones_read (inode_t *`***inode***`,`
`zno_t` ***zone_start***`,`
`void *`***buffer***`,`
`blkcnt_t` ***blkcnt***`);`
	Legge da un file, identificato attraverso il puntatore all'inode (della tabella di inode), una certa quantità di zone, a partire da una certa zona relativa al file, mettendo il risultato della lettura a partire dalla posizione di memoria rappresentata da un puntatore generico. La funzione restituisce la quantità di zone lette con successo.

Funzione	Descrizione

`ssize_t`
`inode_file_read (inode_t *`***inode***`,`
`off_t` ***offset***`,`
`void *`***buffer***`,`
`size_t` ***count***`,`
`int *`***eof***`);`
	Legge il contenuto di un file, individuato da un inode già caricato nella tabella relativa, aggiornando eventualmente una variabile contenente l'indicatore di fine file. La funzione restituisce la quantità di byte letti con successo, oppure il valore `-1` in caso di problemi.

`ssize_t`
`inode_file_write (inode_t *`***inode***`,`
`off_t` ***offset***`,`
`void *`***buffer***`,`
`size_t` ***count***`);`
	Scrive una certa quantità di byte nel file individuato da un inode già caricato nella tabella relativa. La funzione restituisce la quantità di byte scritti effettivamente, oppure il valore `-1` in caso di problemi.

`zno_t inode_zone (inode_t *`***inode***`,`
`zno_t` ***fzone***`,`
`int` ***write***`);`
	Restituisce il numero di zona effettivo, corrispondente a un numero di zona relativo a un certo file di un certo inode. Se il parametro ***write*** è pari a zero, si intende che la zona deve esistere, quindi se questa non c'è, si ottiene semplicemente un valore pari a zero; se invece l'ultimo parametro è pari a uno, nel caso la zona cercata fosse attualmente mancante, verrebbe creata al volo nel file system.

`int` `inode_truncate (inode_t *`***inode***`);`	Riduce la dimensione del file a cui si riferisce l'inode a zero. In pratica fa sì che le zone allocate del file siano liberate. La funzione restituisce zero se l'operazione si conclude con successo, oppure `-1` in caso di problemi.

`int`
`inode_check (inode_t *`***inode***`,`
`mode_t` ***type***`,`
`int` ***perm***`,`
`uid_t` ***uid***`);`
	Verifica che l'inode sia di un certo tipo e abbia i permessi di accesso necessari a un certo utente. Nel parametro ***type*** si possono indicare più tipi validi. La funzione restituisce zero in caso di successo, ovvero di compatibilità, mentre restituisce `-1` se il tipo o i permessi non sono adatti.

`int` `inode_dir_empty (inode_t *`***inode***`);`	Verifica se la directory a cui si riferisce l'inode è effettivamente una directory ed è vuota, nel qual caso restituisce il valore uno, altrimenti restituisce zero.

Fasi dell'innesto di un file system

L'innesto e il distacco di un file system, coinvolge simultaneamente la tabella dei super blocchi e quella degli inode. Si distinguono due situazioni fondamentali: l'innesto del file system principale e quello di un file system ulteriore.

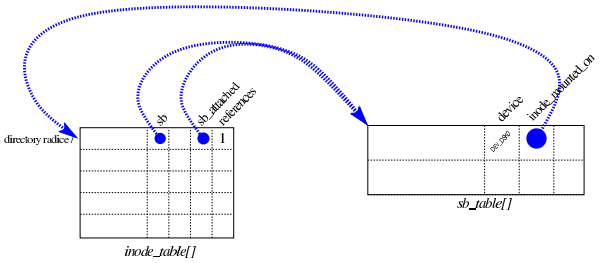
Quando si tratta dell'innesto del file system principale, la tabella dei super blocchi è priva di voci e quella degli inode non contiene

riferimenti a file system. La funzione ***sb_mount()*** viene chiamata indicando, come riferimento all'inode di innesto, il puntatore a una variabile puntatore contenente il valore nullo:

```
…
    inode_t *inode;
    sb_t    *sb;
    …
    inode = NULL;
    sb = sb_mount (DEV_DSK0, &inode, MOUNT_DEFAULT);
    …
```

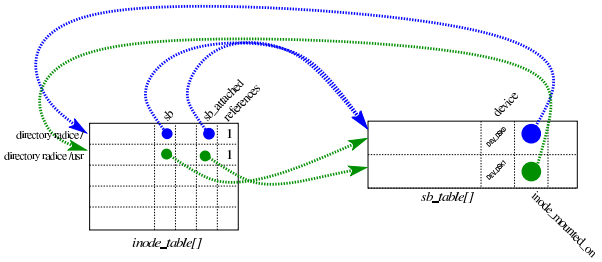
La funzione ***sb_mount()*** carica il super blocco nella tabella relativa, ma trovando il riferimento all'inode di innesto nullo, provvede a caricare l'inode della directory radice dello stesso dispositivo, creando un collegamento incrociato tra le tabelle dei super blocchi e degli inode, come si vede nella figura successiva.

Figura u148.11. Collegamento tra la tabella degli inode e quella dei super blocchi, quando si innesta il file system principale.



Per innestare un altro file system, occorre prima disporre dell'inode di una directory (appropriata) nella tabella degli inode, quindi si può caricare il super blocco del nuovo file system, creando il collegamento tra directory e file system innestato.

Figura u148.12. Innesto di un file system nella directory '`/usr/`'.

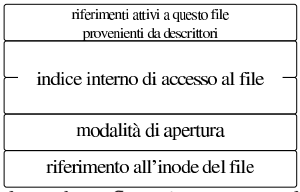


File «kernel/fs/file_...»

I file '`kernel/fs/file_…`' descrivono le funzioni per la gestione della tabella dei file, la quale si collega a sua volta a quella degli inode. In realtà, le funzioni di questo gruppo sono in numero molto limitato, perché l'intervento nella tabella dei file avviene prevalentemente per opera di funzioni che gestiscono i descrittori.

La tabella dei file è rappresentata dall'array ***file_table[]*** e per individuare un certo elemento dell'array si usa preferibilmente la funzione ***file_reference()***. Gli elementi della tabella dei file sono di tipo '**`file_t`**' (definito nel file '`kernel/fs.h`'); una voce della tabella rappresenta un file aperto se il campo dei riferimenti (***references***) ha un valore maggiore di zero.

Figura u148.13. Struttura del tipo '**`file_t`**', corrispondente agli elementi dell'array ***file_table[]***.




```
typedef struct file file_t;

struct file {
  int      references;
  off_t    offset;
  int      oflags;
  inode_t *inode;
};
```

Nel membro ***oflags*** si annotano esclusivamente opzioni relative alla

modalità di apertura del file: lettura, scrittura o entrambe; pertanto si possono usare le macro-variabili ***O_RDONLY***, ***O_WRONLY*** e ***O_RDWR***, come dichiarato nel file di intestazione '`lib/fcntl.h`'. Il membro ***offset*** rappresenta l'indice interno di accesso al file, per l'operazione successiva di lettura o scrittura al suo interno. Il membro ***references*** è un contatore dei riferimenti a questa tabella, da parte di descrittori di file.

La tabella dei file si collega a quella degli inode, attraverso il membro ***inode***. Più voci della tabella dei file possono riferirsi allo stesso inode, perché hanno modalità di accesso differenti, oppure soltanto per poter distinguere l'indice interno di lettura e scrittura. Va osservato che le voci della tabella di inode potrebbero essere usate direttamente e non avere elementi corrispondenti nella tabella dei file.

Figura u148.14. Collegamento tra la tabella dei file e quella degli inode.

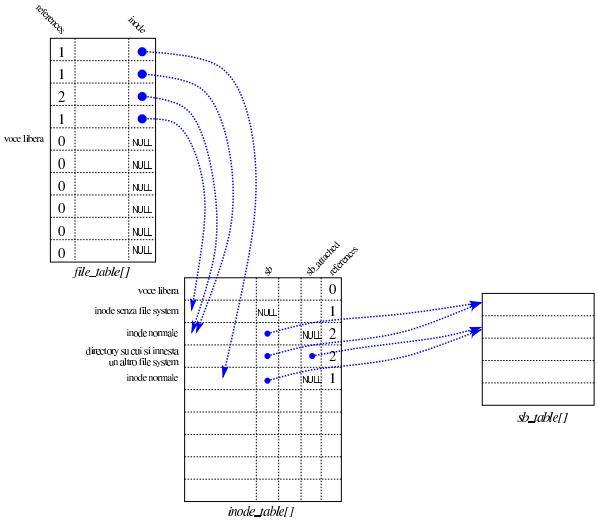


Tabella u148.15. Funzioni fatte esclusivamente per la gestione della tabella dei file ***file_table[]***.

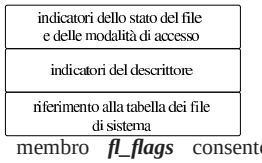
Funzione	Descrizione
`file_t *` `file_reference (int` ***`fno`***`);`	Restituisce il puntatore all'elemento ***fno***-esimo della tabella dei file. Se ***fno*** è un valore negativo, viene restituito il puntatore a una voce libera della tabella.

`file_t *`
`file_stdio_dev_make (dev_t` ***`device`***`,`
`mode_t` ***`mode`***`,`
`int` ***`oflags`***`);`
	Crea una voce per l'accesso a un file di dispositivo standard di input-output, restituendo il puntatore alla voce stessa.

Descrittori di file

Le tabelle di super blocchi, inode e file, riguardano il sistema nel complesso. Tuttavia, l'accesso normale ai file avviene attraverso il concetto di «descrittore», il quale è un file aperto da un certo processo elaborativo. Nel file '`kernel/fs.h`' si trova la dichiarazione e descrizione del tipo derivato '**`fd_t`**', usato per costruire una tabella di descrittori, ma tale tabella non fa parte della gestione del file system, bensì è incorporata nella tabella dei processi elaborativi. Pertanto, ogni processo ha una propria tabella di descrittori di file.

Figura u148.16. Struttura del tipo '**fd_t**', con cui si costituiscono gli elementi delle tabelle dei descrittori di file, una per ogni processo.

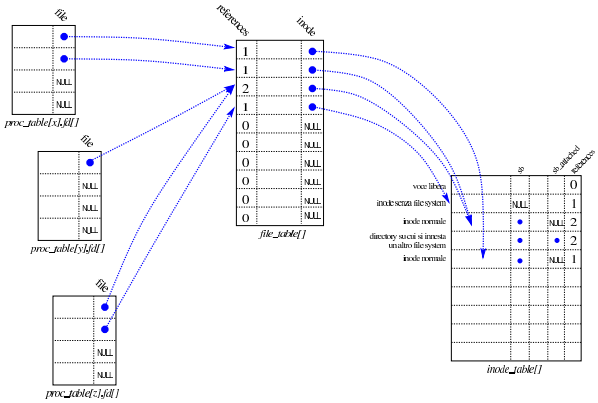



```
typedef struct fd fd_t;
struct fd {
    int     fl_flags;
    int     fd_flags;
    file_t *file;
};
```

Il membro ***fl_flags*** consente di annotare indicatori del tipo '**O_RDONLY**', '**O_WRONLY**', '**O_RDWR**', '**O_CREAT**', '**O_EXCL**', '**O_NOCTTY**', '**O_TRUNC**' e '**O_APPEND**', come dichiarato nella libreria standard, nel file di intestazione '`lib/fcntl.h`'. Tali indicatori si combinano assieme con l'operatore binario OR. Altri tipi di opzione che sarebbero previsti nel file '`lib/fcntl.h`', sono privi di effetto nella gestione del file system di os16.

Il membro ***fd_flags*** serve a contenere, eventualmente, l'opzione '**FD_CLOEXEC**', definita nel file '`lib/fcntl.h`'. Non sono previste altre opzioni di questo tipo.

Figura u148.17. Collegamento tra le tabelle dei descrittori e la tabella complessiva dei file. La tabella ***proc_table[x].fd[]*** rappresenta i descrittori di file del processo elaborativo ***x***.



File «kernel/fs/path_...»

«

I file '`kernel/fs/path_...`' descrivono le funzioni che fanno riferimento a file o directory attraverso una stringa che ne descrive il percorso.

Tabella u148.18. Funzioni per la gestione dei file, a cui si fa riferimento attraverso un percorso, senza indicazioni sul processo elaborativo.

Funzione	Descrizione
`int path_fix (char *path);`	Verifica il percorso indicato semplificandolo, quindi sovrascrive il percorso originario con quello riveduto e corretto. Un percorso assoluto rimane assoluto; un percorso relativo rimane relativo, mancando qualunque indicazione sulla directory corrente.
`int path_full (const char *path, const char *path_cwd, char *full_path);`	Ricostruisce un percorso assoluto, usando come riferimento la directory corrente indicata in ***path_cwd***, salvandolo in ***path_full***.

Tabella u148.19. Funzioni per la gestione dei file, a cui si fa riferimento attraverso un percorso, tenendo conto del processo elaborativo per conto del quale si svolge l'operazione. Del processo elaborativo si considera soprattutto l'identità efficace, per conoscerne i privilegi e determinare se è data effettivamente la facoltà di eseguire l'azione richiesta.

Funzione	Descrizione
`int path_chdir (pid_t pid, const char *path);`	Cambia la directory corrente, utilizzando il nuovo percorso indicato. È l'equivalente della funzione standard ***chdir()*** (sezione u0.3).
`int path_chmod (pid_t pid, const char *path, mode_t mode);`	Cambia la modalità di accesso al file indicato. È l'equivalente della funzione standard ***chmod()*** (sezione u0.4).
`int path_chown (pid_t pid, const char *path, uid_t uid, gid_t gid);`	Cambia l'utente e il gruppo proprietari del file (va però ricordato che os16 non considera i gruppi, anche se nel file system sono annotati). È l'equivalente della funzione standard ***chown()*** (sezione u0.5).
`dev_t path_device (pid_t pid, const char *path);`	Restituisce il numero del dispositivo di un file di dispositivo; pertanto, il percorso deve fare riferimento a un file di dispositivo, per poter ottenere un risultato valido.
`inode_t * path_inode (pid_t pid, const char *path);`	Apre l'inode del file indicato tramite il percorso, purché il processo ***pid*** abbia i permessi di accesso («x») alle directory che vi conducono. La funzione restituisce il puntatore all'inode aperto, oppure il puntatore nullo se non può eseguire l'operazione.
`inode_t *path_inode_link (pid_t pid, const char *path, inode_t *inode, mode_t mode);`	Crea un collegamento fisico con il nome fornito in ***path***, riferito all'inode a cui punta ***inode***, ma se ***inode*** fosse un puntatore nullo, verrebbe semplicemente creato un file vuoto con un nuovo inode. Si richiede inoltre che il processo ***pid*** abbia i permessi di accesso per tutte le directory che portano al file da collegare e che nell'ultima ci sia anche il permesso di scrittura, dovendo intervenire su tale directory in questo modo. Se la funzione riesce nel proprio intento, restituisce il puntatore a ciò che descrive l'inode collegato o creato.
`int path_link (pid_t pid, const char *path_old, const char *path_new);`	Crea un collegamento fisico. È l'equivalente della funzione standard ***link()*** (sezione u0.23).

Funzione	Descrizione
`int path_mkdir (pid_t pid, const char *path, mode_t mode);`	Crea una directory, con la modalità dei permessi indicata. È l'equivalente della funzione standard ***mkdir()*** (sezione u0.25).
`int path_mknod (pid_t pid, const char *path, mode_t mode, dev_t device);`	Crea un file vuoto, con il tipo e i permessi specificati da ***mode***; se si tratta di un file di dispositivo, viene preso in considerazione anche il parametro ***device***, per specificare il numero primario e secondario dello stesso. Va osservato che con questa funzione è possibile creare una directory priva delle voci '.' e '..'. È l'equivalente della funzione standard ***mknod()*** (sezione u0.26).
`int path_stat (pid_t pid, const char *path, struct stat *buffer);`	Aggiorna la variabile strutturata a cui punta ***buffer***, con le informazioni sul file specificato. È l'equivalente della funzione standard ***stat()*** (sezione u0.36).
`int path_unlink (pid_t pid, const char *path);`	Cancella un file o una directory, purché questa sia vuota. È l'equivalente della funzione standard ***unlink()*** (sezione u0.42).
`int path_mount (pid_t pid, const char *path_dev, const char *path_mnt, int options);`	Innesta il dispositivo corrispondente a ***path_dev***, nella directory ***path_mnt*** (tenendo conto della directory corrente del processo ***pid***), con le opzioni specificate, per conto dell'utente ***uid***. Le opzioni disponibili sono solo '**`MOUNT_DEFAULT`**' e '**`MOUNT_RO`**', come dichiarato nel file di intestazione '`lib/sys/os16.h`'.
`int path_umount (pid_t pid, const char *path_mnt);`	Stacca l'unità innestata nella directory indicata, purché nulla al suo interno sia attualmente in uso.

File «kernel/fs/fd_...»

I file '`kernel/fs/fd_...`' descrivono le funzioni che fanno riferimento a file o directory attraverso il numero di descrittore, riferito a sua volta a un certo processo elaborativo. Pertanto, il numero del processo e il numero del descrittore sono i primi due parametri obbligatori di tutte queste funzioni.

Tabella u148.20. Funzioni per la gestione dei file, a cui si fa riferimento attraverso il descrittore, relativamente a un certo processo elaborativo. La funzione ***fd_open()*** fa eccezione, in quanto apre un descrittore, ma per identificare il file non ancora aperto, ne richiede il percorso.

Funzione	Descrizione
`int fd_chmod (pid_t pid, int fdn, mode_t mode);`	Cambia la modalità dei permessi (solo gli ultimi 12 bit del parametro ***mode*** vengono considerati). È l'equivalente della funzione standard ***fchmod()*** (sezione u0.4).

Funzione	Descrizione
`int fd_chown (pid_t pid, int fdn, uid_t uid, gid_t gid);`	Cambia la proprietà (utente e gruppo). È l'equivalente della funzione standard ***fchown()*** (sezione u0.5).
`int fd_close (pid_t pid, int fdn);`	Chiude il descrittore di file. È l'equivalente della funzione standard ***close()*** (sezione u0.7).
`int fd_dup (pid_t pid, int fdn_old, int fdn_min);`	Duplica il descrittore ***fdn_old***, creandone un altro con numero maggiore o uguale a ***fdn_min*** (viene scelto il primo libero a partire da ***fdn_num***). È l'equivalente della funzione standard ***dup()*** (sezione u0.8).
`int fd_dup2 (pid_t pid, int fdn_old, int fdn_new);`	Duplica il descrittore ***fdn_old***, creandone un altro con numero ***fdn_new***. Se però ***fdn_new*** è già aperto, prima della duplicazione questo viene chiuso. È l'equivalente della funzione standard ***dup2()*** (sezione u0.8).
`int fd_fcntl (pid_t pid, int fdn, int cmd, int arg);`	Svolge il compito della funzione standard ***fcntl()*** (sezione u0.13).
`off_t fd_lseek (pid_t pid, int fdn, off_t offset, int whence);`	Riposiziona l'indice interno di accesso del descrittore di file. È l'equivalente della funzione standard ***lseek()*** (sezione u0.24).
`int fd_open (pid_t pid, const char *path, int oflags, mode_t mode);`	Apre un descrittore, fornendo però il percorso del file. È l'equivalente della funzione standard ***open()*** (sezione u0.28).
`ssize_t fd_read (pid_t pid, int fdn, void *buffer, size_t count, int *eof);`	Legge da un descrittore, aggiornando eventualmente la variabile ****eof*** in caso di fine del file. È l'equivalente della funzione standard ***read()*** (sezione u0.29).
`fd_t *fd_reference (pid_t pid, int *fdn);`	Produce il puntatore ai dati del descrittore ****fdn***. Se ****fdn*** è minore di zero, si ottiene il riferimento al primo descrittore libero, aggiornando anche ****fdn*** stesso.
`int fd_stat (pid_t pid, int fdn, struct stat *buffer);`	Svolge il compito della funzione standard ***fstat()*** (sezione u0.36).
`ssize_t fd_write (pid_t pid, int fdn, const void *buffer, size_t count);`	Scrive nel descrittore. È l'equivalente della funzione standard ***write()*** (sezione u0.44).

File «kernel/proc/_isr.s» e «kernel/proc/_ivt_load.s» 1365
Routine «isr_1C» 1368
Routine «isr_80» 1369
La tabella dei processi 1370
Chiamate di sistema 1373
File «kernel/proc/...» 1374
Funzione «proc_init()» 1374
Funzione «sysroutine()» 1374
Funzione «proc_scheduler()» 1375

`proc.h` 1365 `proc_init()` 1374 `proc_reference()` 1374 `proc_scheduler()` 1375 `proc_t` 1370 `sysroutine()` 1373 1374 `_isr.s` 1365 `_ivt_load.s` 1365

La gestione dei processi è raccolta nei file '`kernel/proc.h`' e '`kernel/proc/...`', dove il file '`kernel/proc/_isr.s`', in particolare, contiene il codice attivato dalle interruzioni. Nella semplicità di os16, ci sono solo due interruzioni che vengono gestite: quella del temporizzatore il quale produce un impulso 18,2 volte al secondo, e quella causata dalle chiamate di sistema.

Con os16, quando un processo viene interrotto, per lo svolgimento del compito dell'interruzione, si passa sempre a utilizzare la pila dei dati del kernel. Per annotare la posizione in cui si trova l'indice della pila del kernel si usa la variabile ***_ksp***, accessibile anche dal codice in linguaggio C.

Il codice del kernel può essere interrotto dagli impulsi del temporizzatore, ma in tal caso non viene coinvolto lo schedulatore per lo scambio con un altro processo, così che dopo l'interruzione è sempre il kernel che continua a funzionare; pertanto, nella funzione ***main()*** è il kernel che cede volontariamente il controllo a un altro processo (ammesso che ci sia) con una chiamata di sistema nulla.

File «kernel/proc/_isr.s» e «kernel/proc/_ivt_load.s»

Listati i160.9.1 e i160.9.2.

Il file '`kernel/proc/_isr.s`' contiene il codice per la gestione delle interruzioni dei processi. Nella parte iniziale del file, vengono dichiarate delle variabili, alcune delle quali sono pubbliche e accessibili anche dal codice in C.

```
…
proc_ss_0:       .word 0x0000
proc_sp_0:       .word 0x0000
proc_ss_1:       .word 0x0000
proc_sp_1:       .word 0x0000
proc_syscallnr:  .word 0x0000
proc_msg_offset: .word 0x0000
proc_msg_size:   .word 0x0000
__ksp:           .word 0x0000
__clock_ticks:
ticks_lo:        .word 0x0000
ticks_hi:        .word 0x0000
__clock_seconds:
seconds_lo:      .word 0x0000
seconds_hi:      .word 0x0000
…
```

Si tratta di variabili scalari da 16 bit, tenendo conto che: i simboli '**`ticks_lo`**' e '**`ticks_hi`**' compongono assieme la variabile ***_clock_ticks*** a 32 bit per il linguaggio C; i simboli '**`seconds_lo`**' e '**`seconds_hi`**' compongono assieme la variabile ***_clock_seconds*** a 32 bit per il linguaggio C.

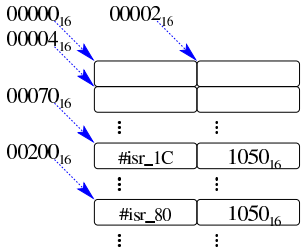
Dopo la dichiarazione delle variabili inizia il codice vero e proprio. Il simbolo '**`isr_1C`**' si riferisce al codice da usare in presenza dell'interruzione $1C_{16}$, mentre il simbolo '**`isr_80`**' riguarda l'interruzione 80_{16}.

Nel file ‘kernel/proc/_ivt_load.s’, la funzione ***_ivt_load()*** che inizia con il simbolo ‘**__ivt_load**’, modifica la tabella IVT (*Interrupt vector table*) in modo che le interruzioni $1C_{16}$ e 80_{16} portino all’esecuzione del codice che inizia rispettivamente in corrispondenza dei simboli ‘**isr_1C**’ e ‘**isr_80**’ (del file ‘kernel/proc/_isr.s’).

```
…
__ivt_load:
    enter #0, #0          ; No local variables.
    pushf
    cli
    pusha
    ;
    mov   ax,   #0        ; Change the DS segment to 0.
    mov   ds,   ax        ;
    ;
    mov   bx,   #112      ; Timer      INT 0x08 (8) --> 0x1C
    mov   [bx], #isr_1C   ; offset
    mov   bx,   #114      ;
    mov   [bx], cs        ; segment
    ;
    mov   bx,   #512      ; Syscall    INT 0x80 (128)
    mov   [bx], #isr_80   ; offset
    mov   bx,   #514      ;
    mov   [bx], cs        ; segment
    ;
    mov   ax,   #0x0050   ; Put the DS segment back to the
    mov   ds,   ax        ; right value.
    ;
    popa
    popf
    leave
    ret
```

Per compiere il suo lavoro, la funzione ***_ivt_load()*** salva inizialmente lo stato degli indicatori contenuti nel registro ***FLAGS*** e gli altri registri principali, quindi modifica il registro ***DS*** in modo che il segmento dati corrisponda allo zero, per poter accedere al contenuto della tabella IVT (che inizia proprio dall’indirizzo 00000_{16}). A quel punto, all’indirizzo efficace 00070_{16} (112_{10}) scrive l’indirizzo relativo del simbolo ‘**isr_1C**’ (l’indirizzo relativo al segmento codice attuale) e il valore del segmento codice all’indirizzo efficace 00072_{16} (114_{10}). Nello stesso modo agisce per il simbolo ‘**isr_80**’, scrivendo il suo indirizzo relativo all’indirizzo efficace 00200_{16} (512_{10}), assieme al valore del segmento codice che va invece in 00202_{16} (514_{10}). In tal modo, quando scatta l’interruzione $1C_{16}$ che deriva dalla scansione del temporizzatore interno, viene eseguito il codice che si trova nella voce corrispondente della tabella IVT, ovvero, proprio ciò che comincia con il simbolo ‘**isr_1C**’, mentre quando scatta l’interruzione 80_{16} si ottiene l’esecuzione del codice che si trova a partire dal simbolo ‘**isr_80**’.

Figura u149.3. Modifica della tabella IVT attraverso la funzione ***_ivt_load()***. Il valore del segmento codice è sicuramente 1050_{16}, in quanto si tratta di quello del kernel, il quale va a collocarsi in quella posizione.



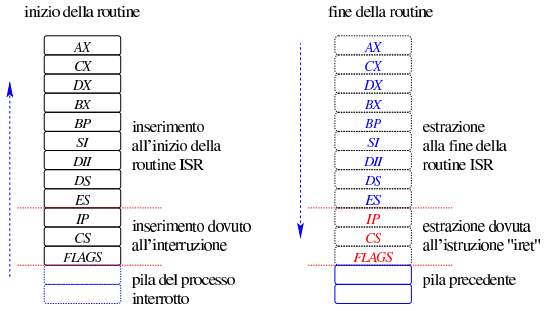
Le interruzioni previste con os16 sono solo due: quella del temporizzatore (*timer*) che invia un impulso a 18,2 Hz circa e quella che serve per le chiamate di sistema. Per la precisione, il temporizzatore fa scattare l’interruzione 08_{16}, ma se si utilizza il codice del BIOS, non può essere ridiretta; pertanto, il codice predefinito per tale interruzione, al termine del suo compito, fa scattare l’interruzione $1C_{16}$, la quale può essere ridiretta come appena mostrato.

Il codice per le due interruzioni gestite è simile, con la differenza fondamentale che per l’interruzione proveniente dal temporizzatore si incrementano i contatori rappresentati dalle variabili ***_clock_ticks*** e ***_clock_seconds***. Il codice equivalente della gestione delle due interruzioni è il seguente:

```
…
isr_1C: isr_80:
    push   es   ; extra segment
    push   ds   ; data segment
    push   di   ; destination index
    push   si   ; source index
    push   bp   ; base pointer
    push   bx   ; BX
    push   dx   ; DX
    push   cx   ; CX
    push   ax   ; AX
    ;
    mov ax, #0x0050  ; DS and ES.
    mov ds, ax       ;
    mov es, ax       ;
    …
    …
    pop    ax
    pop    cx
    pop    dx
    pop    bx
    pop    bp
    pop    si
    pop    di
    pop    ds
    pop    es
    ;
    iret
…
```

Mentre viene eseguito il codice che si trova a partire da ‘**isr_1C**’ o da ‘**isr_80**’, il segmento codice è quello del kernel, ma quello dei dati è quello del processo che è stato interrotto poco prima. Nella pila dei dati di quel processo, nel momento in cui viene raggiunto questo codice ci sono già i valori di alcuni registri, nello stato in cui erano al verificarsi dell’interruzione: ***FLAGS***, ***CS***, ***IP***. Come si vede dal codice appena mostrato, si aggiungono nella pila altri registri.

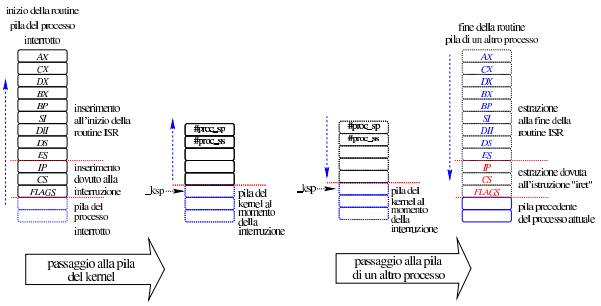
Figura u149.5. Inserimento nella pila del processo interrotto.



Dopo il salvataggio nella pila dei registri principali, viene modificato il valore dei registri ***DS*** e ***ES***, per consentire l’accesso alle variabili dichiarate all’inizio del file ‘kernel/_isr.s’. Il valore che si attribuisce a tali registri è 0050_{16}, perché il segmento dati del kernel inizia all’indirizzo efficace 00500_{16}. Va osservato che il segmento usato per la pila dei dati non viene ancora modificato e rimane nel segmento dati del processo interrotto.

A questo punto iniziano le differenze tra le due routine di gestione delle interruzioni. In ogni caso rimane il principio di massima, descritto intuitivamente dalla figura successiva, per cui si scambia la pila del processo interrotto con quella del kernel, poi si esegue la chiamata di sistema o si attiva lo schedulatore, quindi si passa nuovamente alla pila di un processo, il quale può essere diverso da quello interrotto.

Figura u149.6. Scambi delle pile.



Routine «isr_1C»

Dopo il salvataggio dei registri principali e dopo il cambiamento del segmento dati, rimanendo ancora sulla pila dei dati del processo interrotto, la routine '**isr_1C**' si occupa di incrementare i contatori degli impulsi e dei secondi:

```
…
isr_1C:
    …
    add ticks_lo, #1    ; Clock ticks counter.
    adc ticks_hi, #0    ;
    ;
    mov dx, ticks_hi    ;
    mov ax, ticks_lo    ; DX := ticks % 18
    mov cx, #18         ;
    div cx              ;
    mov ax, #0          ; If the ticks value can be divided
    cmp ax, dx          ; by 18, the seconds is incremented
    jnz L1              ; by 1.
    add seconds_lo, #1  ;
    adc seconds_hi, #0  ;
    ;
L1:
    …
```

Per semplificare i calcoli, si considera che ogni 18 impulsi sia trascorso un secondo e di conseguenza va interpretata la divisione che viene eseguita. In ogni caso, quando si arriva al simbolo '**L1**' le variabili sono state aggiornate correttamente.

A questo punto viene salvato il valore del segmento in cui si trova la pila dei dati e l'indice all'interno della stessa, usando delle variabili locali, le quali non sono però accessibili dal codice in linguaggio C:

```
…
L1:
    mov proc_ss_0, ss   ; Save process stack segment.
    mov proc_sp_0, sp   ; Save process stack pointer.
    …
```

Poi si verifica se la pila dei dati del processo interrotto si trova nel kernel. In tal caso, il suo segmento avrebbe il valore 0050_{16}. Se il segmento dati è proprio quello del kernel, si saltano le istruzioni successive, riprendendo dal ripristino dei registri dalla pila dei dati (dal simbolo '**L2**').

```
    …
    mov dx, proc_ss_0
    mov ax, #0x0050     ; Kernel data area.
    cmp dx, ax
    je L2
    …
```

Se non è il kernel che è stato interrotto, si fa in modo di saltare all'utilizzo della pila dei dati del kernel. Per fare questo viene sostituito il valore del registro '**SS**', facendo in modo che corrisponda al segmento dati del kernel stesso, quindi si modifica il valore del registro '**SP**', mettendovi il valore salvato precedentemente nella variabile ***_ksp*** (ovvero il simbolo '**__ksp**').

```
    …
    mov ax, #0x0050     ; Kernel data area.
    mov ss, ax
    mov sp, __ksp
    …
```

Nella variabile ***_ksp*** c'è sicuramente l'indice della pila del kernel, aggiornata dalla funzione ***proc_scheduler()***. Tale aggiornamento della variabile ***_ksp*** avviene quando il gestore dei processi elaborativi sospende il codice del kernel per mettere in funzione un altro processo.

A questo punto, il contesto esecutivo è diventato quello del kernel, provenendo però dall'interruzione di un altro processo. Quindi viene chiamata la funzione di attivazione dello schedulatore: ***proc_scheduler()***. Tale funzione richiede dei parametri e gli vengono forniti i puntatori alle variabili contenenti il segmento e l'indice della pila dei dati del processo interrotto.

```
    …
    push #proc_ss_0     ; &proc_ss_0
    push #proc_sp_0     ; &proc_sp_0
    call _proc_scheduler
    add  sp, #2
    add  sp, #2
    …
```

Al termine del lavoro della funzione ***proc_scheduler()***, i valori contenuti nelle variabili rappresentate dai simboli '**proc_ss_0**' e '**proc_sp_0**' possono essere stati sostituiti con quelli di un altro processo da attivare al posto di quello interrotto precedentemente. Infatti, i registri ***SS*** e ***SP*** vengono sostituiti subito dopo:

```
    …
    mov ss, proc_ss_0   ; Restore process stack segment.
    mov sp, proc_sp_0   ; Restore process stack pointer.
    …
```

Infine, si ripristinano gli altri registri, traendo i dati dalla nuova pila.

Routine «isr_80»

Dopo il salvataggio dei registri principali e dopo il cambiamento del segmento dati, rimanendo ancora sulla pila dei dati del processo interrotto, la routine '**isr_80**' salva il valore del segmento in cui si trova la pila dei dati e l'indice all'interno della stessa, usando delle variabili locali, le quali non sono però accessibili dal codice in C:

```
    …
    mov proc_ss_1, ss   ; Save process stack segment.
    mov proc_sp_1, sp   ; Save process stack pointer.
    …
```

Vengono quindi salvati dei dati contenuti ancora nella pila attuale, utilizzando delle variabili statiche, che però non sono accessibili dal codice C:

```
    …
    mov bp, sp
    mov ax, +26[bp]
    mov proc_syscallnr,  ax
    mov ax, +28[bp]
    mov proc_msg_offset, ax
    mov ax, +30[bp]
    mov proc_msg_size,   ax
    …
```

Finalmente si passa a verificare se il processo interrotto è il kernel o meno: se si tratta proprio del kernel, il valore del registro ***SP*** viene salvato nella variabile ***_ksp***.

```
    …
    mov dx, ss
    mov ax, #0x0050     ; Kernel data area.
    cmp dx, ax
    jne L3
    mov __ksp, sp
L3:
```

```
    …
```

Successivamente si scambia la pila dei dati attuale, passando a quella del kernel, utilizzando la variabile ***_ksp*** per modificare il registro ***SP***. Naturalmente si comprende che se il codice interrotto era già quello del kernel, la sostituzione non cambia in pratica i valori che già avevano i registri ***SS*** e ***SP***:

```
    …
L3:
    mov ax, #0x0050     ; Kernel data area.
    mov ss, ax
    mov sp, __ksp
    …
```

Quando la pila dei dati in funzione è quella del kernel, si passa alla chiamata della funzione ***sysroutine()***, passandole come parametri i dati raccolti precedentemente dalla pila del processo interrotto, fornendo anche i puntatori alle variabili che contengono i dati necessari a raggiungere tale pila.

```
    …
    push proc_msg_size
    push proc_msg_offset
    push proc_syscallnr
    push #proc_ss_1     ; &proc_ss_1
    push #proc_sp_1     ; &proc_sp_1
    call _sysroutine
    add  sp, #2
    add  sp, #2
    add  sp, #2
    add  sp, #2
    add  sp, #2
    …
```

La funzione ***sysroutine()*** chiama a sua volta la funzione ***proc_scheduler()***, la quale può modificare il contenuto delle variabili rappresentate dai simboli '**`proc_ss_1`**' e '**`proc_sp_1`**'; pertanto, quando i valori di tali variabili vengono usati per rimpiazzare il contenuto dei registri ***SS*** e ***SP***, si ottiene lo scambio a un processo diverso da quello interrotto inizialmente.

```
    …
    mov ss, proc_ss_1   ; Restore process stack segment.
    mov sp, proc_sp_1   ; Restore process stack pointer.
    …
```

Infine, si ripristinano gli altri registri, traendo i dati dalla nuova pila.

La tabella dei processi

«

Listato u0.9.

Nel file '`kernel/proc.h`' viene definito il tipo '**`proc_t`**', con il quale, nel file '`kernel/proc/proc_table.c`' si definisce la tabella dei processi, rappresentata dall'array ***proc_table[]***.

Figura u149.19. Struttura del tipo '**`proc_t`**', corrispondente agli elementi dell'array ***proc_table[]***.

Listato u149.20. Struttura del tipo '**`proc_t`**', corrispondente agli elementi dell'array ***proc_table[]***.

```
typedef struct {
  pid_t             ppid;
  pid_t             pgrp;
  uid_t             uid;
  uid_t             euid;
  uid_t             suid;
  dev_t             device_tty;
  char              path_cwd[PATH_MAX];
  inode_t          *inode_cwd;
  int               umask;
  unsigned long int sig_status;
  unsigned long int sig_ignore;
  clock_t           usage;
  unsigned int      status;
  int               wakeup_events;
  int               wakeup_signal;
  unsigned int      wakeup_timer;
  addr_t            address_i;
  segment_t         segment_i;
  size_t            size_i;
  addr_t            address_d;
  segment_t         segment_d;
  size_t            size_d;
  uint16_t          sp;
  int               ret;
  char              name[PATH_MAX];
  fd_t              fd[FOPEN_MAX];
} proc_t;
```

La tabella successiva descrive il significato dei vari membri previsti dal tipo '**`proc_t`**'. Va osservato che os16 non gestisce i gruppi di utenti, anche se questi sono previsti comunque nel file system, pertanto la tabella dei processi è più semplice rispetto a quella di un sistema conforme allo standard di Unix. Un'altra considerazione va fatta a proposito della cosiddetta «u-area» (*user area*), la quale non viene gestita come un sistema Unix tradizionale e tutti i dati dei processi sono raccolti nella tabella gestita dal kernel. Di conseguenza, dal momento che i processi non dispongono di una tabella personale con i dati della u-area, devono avvalersi sempre di chiamate di sistema per leggere i dati del proprio processo.

Tabella u149.21. Membri del tipo '**`proc_t`**'.

Membro	Contenuto
`ppid`	Numero del processo genitore: *parent process id*.

Membro	Contenuto
`pgrp`	Numero del gruppo di processi a cui appartiene quello della voce corrispondente: *process group*. Si tratta del numero del processo a partire dal quale viene definito il gruppo.
`uid`	Identità reale del processo della voce corrispondente: *user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è stato avviato. Tuttavia, i privilegi del processo dipendono dall'identità efficace, definita dal membro '**`euid`**'.
`euid`	Identità efficace del processo della voce corrispondente: *effective user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è in funzione; pertanto, il processo ha i privilegi di quell'utente.
`suid`	Identità salvata: *saved user id*. Si tratta del valore che aveva ***euid*** prima di cambiare identità.
`device_tty`	Terminale di controllo, espresso attraverso il numero del dispositivo.
`path_cwd` `inode_cwd`	Entrambi i membri rappresentano la directory corrente del processo: nel primo caso in forma di percorso, ovvero di stringa, nel secondo in forma di puntatore a inode rappresentato in memoria.
`umask`	Maschera dei permessi associata al processo: i permessi attivi nella maschera vengono tolti in fase di creazione di un file o di una directory.
`sig_status`	Segnali inviati al processo e non ancora trattati: ogni segnale si associa a un bit differente del valore del membro ***sig_status***; un bit a uno indica che il segnale corrispondente è stato ricevuto e non ancora trattato.
`sig_ignore`	Segnali che il processo ignora: ogni segnale da ignorare si associa a un bit differente del valore del membro ***sig_ignore***; un bit a uno indica che quel segnale va ignorato.
`usage`	Tempo di utilizzo della CPU, da parte del processo, espresso in impulsi del temporizzatore, il quale li produce alla frequenza di circa 18,2 Hz.
`status`	Stato del processo, rappresentabile attraverso una macro-variabile simbolica, definita nel file '`proc.h`'. Per os16, gli stati possibili sono: «inesistente», quando si tratta di una voce libera della tabella dei processi; «creato», quando un processo è appena stato creato; «pronto», quando un processo è pronto per essere eseguito, «in esecuzione», quando il processo è in funzione; «sleeping», quando un processo è in attesa di qualche evento; «zombie», quando un processo si è concluso, ha liberato la memoria, ma rimangono le sue tracce perché il genitore non ha ancora recepito la sua fine.
`wakeup_events`	Eventi attesi per il risveglio del processo, ammesso che si trovi nello stato si attesa. Ogni tipo di evento che può essere atteso corrisponde a un bit e si rappresenta con una macro-variabile simbolica, dichiarata nel file '`lib/sys/os16.h`'.
`wakeup_signal`	Ammesso che il processo sia in attesa di un segnale, questo membro esprime il numero del segnale atteso.
`wakeup_timer`	Ammesso che il processo sia in attesa dello scadere di un conto alla rovescia, questo membro esprime il numero di secondi che devono ancora trascorrere.

Membro	Contenuto

`address_i`
`segment_i`
`size_i`
	Il valore di questi membri descrive la memoria utilizzata dal processo per le istruzioni (il segmento codice). Le informazioni sono in parte ridondanti, perché conoscendo ***segment_i*** si ottiene facilmente ***address_i*** e viceversa, ma ciò consente di ridurre i calcoli nelle funzioni che ne fanno uso.

`address_d`
`segment_d`
`size_d`
	Il valore di questi membri descrive la memoria utilizzata dal processo per i dati (il segmento usato per le variabili statiche e per la pila). Anche in questo caso, le informazioni sono in parte ridondanti, ma ciò consente di semplificare il codice nelle funzioni che ne fanno uso.

`sp`	Indice della pila dei dati, nell'ambito del segmento dati del processo. Il valore è significativo quando il processo è nello stato di pronto o di attesa di un evento. Quando invece un processo era attivo e viene interrotto, questo valore viene aggiornato.
`ret`	Rappresenta il valore restituito da un processo terminato e passato nello stato di «zombie».
`name`	Il nome del processo, rappresentato dal nome del programma avviato.
`fd`	Tabella dei descrittori dei file relativi al processo.

Chiamate di sistema

I processi eseguono una chiamata di sistema attraverso la funzione ***sys()***, dichiarata nel file '`lib/sys/os16/sys.s`'. La funzione in sé, per come è dichiarata, potrebbe avere qualunque parametro, ma in pratica ci si attende che il suo prototipo sia il seguente:

```
void sys (syscallnr, void *message, size_t size);
```

Il numero della chiamata di sistema, richiesto come primo parametro, si rappresenta attraverso una macro-variabile simbolica, definita nel file '`lib/sys/os16.h`'.

Per fornire dei dati a quella parte di codice che deve svolgere il compito richiesto, si usa una variabile strutturata, di cui viene trasmesso il puntatore (riferito al segmento dati del processo che esegue la chiamata) e la dimensione complessiva.

Nel file '`lib/sys/os16.h`' sono definiti dei tipi derivati, riferiti a variabili strutturate, per ogni tipo di chiamata. Per esempio, per la chiamata di sistema usata per cambiare la directory corrente del processo, si usa un messaggio di tipo '**`sysmsg_chdir_t`**':

```
typedef struct {
    char          path[PATH_MAX];
    int           ret;
    int           errno;
    int           errln;
    char          errfn[PATH_MAX];
} sysmsg_chdir_t;
```

In realtà, la funzione ***sys()***, si limita a produrre un'interruzione software, da cui viene attivata la routine che inizia al simbolo '**`isr_80`**' nel file '`kernel/_isr.s`', la quale estrapola le informazioni salienti dalla pila dei dati e poi le fornisce alla funzione ***sysroutine()***:

```
void sysroutine (uint16_t *sp, segment_t *segment_d,
                 uint16_t syscallnr, uint16_t msg_off,
                 uint16_t msg_size);
```

Nella funzione ***sysroutine()***, gli ultimi tre parametri corrispondono in pratica agli argomenti della chiamata della funzione ***sys()***, con la differenza che nei vari passaggi hanno perso l'identità originaria e

giungono come numeri puri e semplici, secondo la «parola» del tipo di architettura utilizzato.

File «kernel/proc/...»

Listati successivi a u0.9.

Nella directory '`kernel/proc/`' si trovano i file che realizzano le funzioni dichiarate all'interno di '`kernel/proc.h`'.

Nella gestione dei processi entrano in gioco due variabili globali importanti: ***_ksp*** e ***_etext***. La prima è dichiarata nel file '`kernel/_isr.s`' e viene utilizzata per annotare l'indice della pila dei dati del kernel; la seconda è dichiarata implicitamente dal collegatore (*linker*) e contiene la dimensione dell'area occupata in memoria dal codice del kernel stesso.

Nel file '`kernel/proc/proc_table.c`' è dichiarata la tabella dei processi, attraverso un array composto da elementi di tipo '**`proc_t`**'. La quantità di elementi di questo array costituisce il limite alla quantità di processi gestibili simultaneamente, incluso il kernel e i processi zombie.

Per accedere uniformemente al contenuto della tabella, si usa la funzione ***proc_reference()***, la quale, con l'indicazione del numero del processo (PID), restituisce il puntatore all'elemento della tabella che contiene i dati dello stesso.

Nelle sezioni successive si descrivono solo le funzioni principali della directory '`kernel/proc/`'.

Funzione «proc_init()»

```
void proc_init (void);
```

La funzione ***proc_init()*** viene chiamata dalla funzione ***main()***, una volta sola, per attivare la gestione dei processi elaborativi. Si occupa di compiere le azioni seguenti:

- modificare la tabella delle interruzioni (IVT), attraverso la chiamata della funzione ***_ivt_load()*** (per comodità si usa la macroistruzione ***ivt_load()***), dichiarata nel file '`kernel/proc/_ivt_load.s`';
- impostare la frequenza del temporizzatore, ma tale frequenza deve essere obbligatoriamente di 18,2 Hz;
- azzerare la tabella dei processi;
- innestare il file system principale;
- assegnare i valori appropriati alla voce della tabella dei processi che si riferisce al kernel (PID zero);
- allocare la memoria già utilizzata dal kernel e lo spazio che va da zero fino a 00500_{16} (tabella IVT e BDA);
- attivare selettivamente le interruzioni hardware desiderate.

Funzione «sysroutine()»

La funzione ***sysroutine()*** viene chiamata esclusivamente dalla routine attivata dalle chiamate di sistema (tale routine è introdotta dal simbolo '**`isr_80`**' nel file '`kernel/proc/_isr.s`') e ha una serie di parametri, come si può vedere dal prototipo:

```
void sysroutine (uint16_t *sp, segment_t *segment_d,
                 uint16_t syscallnr, uint16_t msg_off,
                 uint16_t msg_size);
```

I primi due parametri della funzione sono puntatori a variabili dichiarate nel file '`kernel/proc/_isr.s`'. La prima delle due variabili è l'indice della pila dei dati del processo che ha eseguito la chiamata di sistema; la seconda contiene l'indirizzo del segmento dati di tale processo. Il valore del segmento dati serve a individuare il processo elaborativo nella tabella dei processi, dal momento che con os16 i dati non sono condivisibili tra processi.

Il terzo parametro è il numero della chiamata di sistema che ha provocato l'interruzione. Gli ultimi due parametri danno la posizione e la dimensione del messaggio inviato attraverso la chiamata di sistema.

All'inizio della funzione viene individuato il processo elaborativo corrispondente a quello che utilizza il segmento dati ****segment_d*** e l'indirizzo efficace dell'area di memoria contenente il messaggio della chiamata di sistema:

```
    pid_t  pid      = proc_find (*segment_d);
    addr_t msg_addr = address (*segment_d, msg_off);
```

Quindi viene dichiarata un'unione di variabili strutturate, corrispondente alla sovrapposizione di tutti i tipi di messaggio gestibili:

```
    union {
        sysmsg_chdir_t     chdir;
        sysmsg_chmod_t     chmod;
        …
    } msg;
```

A questo punto si verifica se il processo interrotto dalla sua chiamata di sistema è il kernel, perché al kernel è consentito di eseguire solo alcuni tipi di chiamata e tutto il resto sarebbe un errore.

Proseguendo con il codice si vede l'uso della funzione ***dev_io()***, con la quale si legge il messaggio della chiamata di sistema, dalla sua collocazione originale, in un'area tampone del segmento dati del kernel:

```
    dev_io (pid, DEV_MEM, DEV_READ, msg_addr, &msg,
            msg_size, NULL);
```

A questo punto, sapendo di quale chiamata di sistema si tratta, il messaggio può essere letto come:

```
msg.tipo_chiamata
```

Per esempio, per la chiamata di sistema '**`SYS_CHDIR`**', si deve fare riferimento al messaggio ***msg.chdir***; pertanto, per raggiungere il membro ***ret*** del messaggio si usa la notazione ***msg.chdir.ret***.

Una volta eseguita una copia del messaggio, con la funzione ***dev_io()***, si passa a una struttura di selezione, con cui si eseguono operazioni differenti in base al tipo di chiamata ricevuta:

```
    switch (syscallnr)
      {
        case SYS_0:
            break;
        case SYS_CHDIR:
            msg.chdir.ret = path_chdir (pid, msg.chdir.path);
            sysroutine_error_back (&msg.chdir.errno,
                                   &msg.chdir.errln,
                                   msg.chdir.errfn);
            break;
        …
      }
```

Il messaggio usato per trasmettere i dati della chiamata, può servire anche per restituire dei dati al mittente, pertanto, spesso alcuni contenuti dello stesso vengono modificati. Ciò succede particolarmente con il membro ***ret*** che generalmente rappresenta il valore restituito dalla chiamata di sistema. Per questa ragione, dopo la struttura di selezione si ricopia nuovamente il messaggio nella posizione di partenza:

```
    dev_io (pid, DEV_MEM, DEV_WRITE, msg_addr, &msg,
            msg_size, NULL);
```

Al termine del lavoro, viene chiamata la funzione ***proc_scheduler()***.

Funzione «proc_scheduler()»

La funzione ***proc_scheduler()*** richiede come parametri due puntatori: il primo parametro deve essere il riferimento a un valore che rappresenta l'indice della pila di quel processo; il secondo parametro si riferisce a una variabile contenente il valore del segmento dati

del processo interrotto. La funzione richiede queste informazioni in forma di puntatore, per poter modificare i valori delle variabili relative, in modo da consentire così l'attivazione successiva di un altro processo, al posto di quello da cui si proviene.

```
void proc_scheduler (uint16_t *sp, segment_t *segment_d);
```

Inizialmente, la funzione acquisisce il numero del processo interrotto:

```
    prev = proc_find (*segment_d);
```

Quindi svolge delle operazioni che riguardano tutti i processi: aggiorna i contatori dei processi che attendono lo scadere di un certo tempo; verifica la presenza di segnali e predispone le azioni relative; raccoglie l'input dai terminali.

```
    proc_sch_timers ();
    …
    proc_sch_signals ();
    …
    proc_sch_terminals ();
```

A quel punto aggiorna il tempo di utilizzo della CPU del processo appena interrotto:

```
    current_clock   = k_clock ();
    ps[prev].usage += current_clock - previous_clock;
    previous_clock  = current_clock;
```

Quindi inizia la ricerca di un altro processo, candidato a essere ripreso, al posto di quello interrotto. La ricerca inizia dal processo successivo a quello interrotto, senza considerare alcun criterio di precedenza. Il ciclo termina se la ricerca incontra di nuovo il processo di partenza.

```
    for (next = prev+1; next != prev; next++)
      {
        if (next >= PROCESS_MAX)
          {
            next = -1; // At the next loop, 'next' will be
                       // zero.
            continue;
          }
        …
      }
```

All'interno di questo ciclo di ricerca, se si incontra un processo pronto per essere messo in funzione, lo si scambia con quello interrotto: in pratica si salva il valore attuale dell'indice della pila, si scambiano gli stati e si aggiornano i valori di ***sp** e ***segment_d**, in modo da ottenere effettivamente lo scambio all'uscita dalla funzione:

```
        else if (ps[next].status == PROC_READY)
          {
            if (ps[prev].status == PROC_RUNNING)
              {
                ps[prev].status = PROC_READY;
              }
            ps[prev].sp     = *sp;
            ps[next].status = PROC_RUNNING;
            ps[next].ret    = 0;
            *segment_d      = ps[next].segment_d;
            *sp             = ps[next].sp;
            break;
          }
```

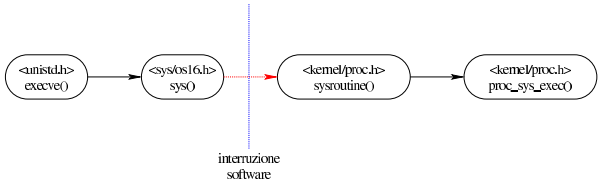
Alla fine del ciclo, occorre verificare se esiste effettivamente un processo successivo attivato, perché in caso contrario, si lascia il controllo direttamente al kernel. In fine, si salva il valore accumulato in precedenza dell'indice della pila del kernel, nella variabile ***_ksp***, quindi si manda il messaggio EOI al circuito del PIC (*programmable interrupt controller*), diversamente non ci sarebbero più, altre interruzioni.

Caricamento ed esecuzione delle applicazioni

Caricamento in memoria 1377
Il codice iniziale dell'applicativo 1378

Caricare un programma e metterlo in esecuzione è un processo delicato che parte dalla funzione ***execve()*** della libreria standard e viene svolto dalla funzione ***proc_sys_exec()*** del kernel.

Figura u150.1. Da ***execve()*** a ***proc_sys_exec()***.



Caricamento in memoria

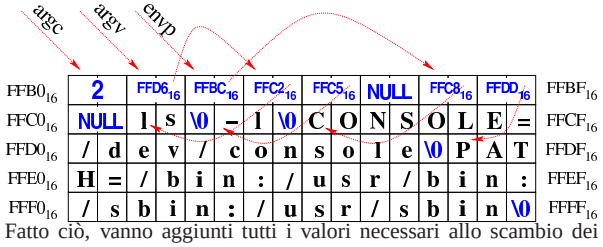
La funzione ***proc_sys_exec()*** (listato i160.9.21) del kernel è quella che svolge il compito di caricare un processo in memoria e di annotarlo nella tabella dei processi.

La funzione, dopo aver verificato che si tratti di un file eseguibile valido e che ci siano i permessi per metterlo in funzione, procede all'allocazione della memoria, dividendo se necessario l'area codice da quella dei dati, quindi legge il file e copia opportunamente le componenti di questo nelle aree di memoria allocate.

La realizzazione attuale della funzione ***proc_sys_exec()*** non è in grado di verificare se un processo uguale sia già in memoria, quindi carica la parte del codice anche se questa potrebbe essere già disponibile.

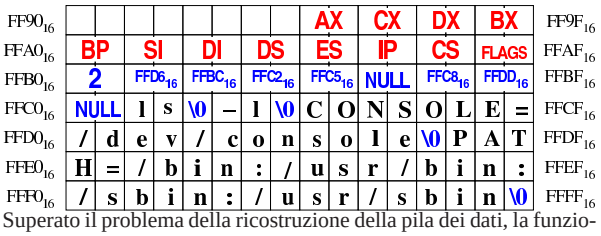
Terminato il caricamento del file, viene ricostruita in memoria la pila dei dati del processo. Prima si mettono sul fondo le stringhe delle variabili di ambiente e quelle degli argomenti della chiamata, quindi si aggiungono i puntatori alle stringhe delle variabili di ambiente, ricostruendo così l'array noto convenzionalmente come '**`envp[]`**', continuando con l'aggiunta dei puntatori alle stringhe degli argomenti della chiamata, per riprodurre l'array '**`argv[]`**'. Per ricostruire gli argomenti della chiamata della funzione ***main()*** dell'applicazione, vanno però aggiunti ancora: il puntatore all'inizio dell'array delle stringhe che descrivono le variabili di ambiente, il puntatore all'array delle stringhe che descrivono gli argomenti della chiamata e il valore che rappresenta la quantità di argomenti della chiamata.

Figura u150.2. Caricamento degli argomenti della chiamata della funzione ***main()***.



Fatto ciò, vanno aggiunti tutti i valori necessari allo scambio dei processi, costituiti dai vari registri da rimpiazzare.

Figura u150.3. Completamento della pila con i valori dei registri.



Superato il problema della ricostruzione della pila dei dati, la funzione ***proc_sys_exec()*** predispone i descrittori di standard input, standard output e standard error, quindi libera la memoria usata dal processo chiamante e ne rimpiazza i dati nella tabella dei processi con quelli del nuovo processo caricato.

Il codice iniziale dell'applicativo

I programmi iniziano con il codice che si trova nel file '`applic/crt0.s`'. Questo file ha delle affinità con il file '`kernel/main/crt0.s`' del kernel, dove la prima differenza che si incontra riguarda l'impronta di riconoscimento. A parte questo, va considerato che il codice delle applicazioni viene eseguito in un momento in cui i registri di segmento sono già stati impostati e l'indice della pila è già collocato correttamente; inoltre, se la funzione ***main()*** termina e restituisce il controllo a '`crt0.s`', un ciclo senza fine esegue continuamente una chiamata di sistema per la conclusione del processo elaborativo corrispondente.

Figura u150.4. Codice iniziale degli applicativi e variabile strutturata di tipo '**`header_t`**'.

```
entry startup
.text
startup:
    jmp startup_code
filler:
    .space (0x0004 - (filler - startup))
magic:
    .data4 0x6F733136
    .data4 0x6170706C
segoff:
    .data2 __segoff
etext:
    .data2 __etext
edata:
    .data2 __edata
ebss:
    .data2 __end
stack_size:
    .data2 0x2000
.align 2
startup_code:
…
```

```
typedef struct {
    uint32_t filler0;
    uint32_t magic0;
    uint32_t magic1;
    uint16_t segoff;
    uint16_t etext;
    uint16_t edata;
    uint16_t ebss;
    uint16_t ssize;
} header_t;
```

La figura mostra il confronto tra il codice iniziale contenuto nel file '`applic/crt0.s`', senza preamboli e senza commenti, con la dichiarazione del tipo derivato '**`header_t`**', presente nel file '`kernel/proc.h`'. Attraverso questa struttura, la funzione ***proc_sys_exec()*** è in grado di estrapolare dal file le informazioni necessarie a caricarlo correttamente in memoria.

Come già accennato, quando viene eseguito il codice di un programma applicativo, la pila dei dati è già operativa. Pertanto, dopo il simbolo '**`startup_code`**' si può già lavorare con questa.

```
startup_code:
    pop ax              ; argc
    pop bx              ; argv
    pop cx              ; envp
    mov _environ, cx    ; Variable 'environ' comes from
                        ; 'unistd.h'.
    push cx
    push bx
    push ax
```

Per prima cosa, viene estratto dalla pila il puntatore all'array noto come ***envp[]***, per poter assegnare tale valore alla variabile ***environ***, come richiede lo standard della libreria POSIX. Tuttavia, per poter gestire poi le variabili di ambiente, si rende necessario utilizzare un array più «comodo», quando le stringhe vanno sostituite. A tale proposito, nel file '`lib/stdlib/environment.c`', si dichiarano ***_environment_table[][]*** e ***_environment[]***. Il primo è semplicemente un array di caratteri, dove, utilizzando due indici di accesso, si conviene di allocare delle stringhe, con una dimensione massima prestabilita. Il secondo, invece, è un array di puntatori, per localizzare l'inizio delle stringhe contenute nel primo. In pratica, alla fine ***_environment[]*** e ***environ[]*** devono essere equivalenti. Ma per attuare questo, occorre utilizzare la funzione ***_environment_setup()*** che sistema tutti i puntatori necessari.

```
    push cx
    call __environment_setup
    add  sp, #2
    ;
    mov  ax, #__environment
    mov  _environ, ax
    ;
    pop ax              ; argc
    pop bx              ; argv[][]
    pop cx              ; envp[][]
    mov cx, #__environment
    push cx
    push bx
    push ax
```

Come si vede dall'estratto del file '`applic/crt0.s`', si vede l'uso della funzione ***_environment_setup()*** (il registro CX contiene già il puntatore a ***envp[]***, e viene inserito nella pila proprio come argomento per la funzione). Successivamente viene riassegnata anche la variabile ***environ*** in modo da coincidere con ***_environment***. Alla fine, viene ricostruita la pila per gli argomenti della chiamata della funzione ***main()***, ma prima di procedere con quella chiamata, si utilizzano due funzioni, per inizializzare la gestione dei flussi di file e delle directory, sempre in forma di flussi.

```
    call __stdio_stream_setup
    call __dirent_directory_stream_setup
    ;
    call _main
    ;
    mov  exit_value, ax
    …
.align 2
.data
exit_value:
    .data2 0x0000
.align 2
.bss
```

La funzione ***_stdio_stream_setup()***, contenuta nel file '`lib/stdio/FILE.c`', associa i descrittori standard ai flussi di file standard (standard input, standard output e standard error); la funzione ***_dirent_directory_stream_setup()*** compie un lavoro analogo, limitandosi però a inizializzare un array di flussi di directory.

Dopo queste preparazioni, viene chiamata la funzione ***main()***, la quale riceve regolarmente i propri argomenti previsti. Il valore restituito dalla funzione viene poi salvato in corrispondenza del simbolo '**`exit_value`**'.

```
halt:
    push #2             ; Size of message.
    push #exit_value    ; Pointer to the message.
    push #6             ; SYS_EXIT
    call _sys
    add  sp, #2
    add  sp, #2
    add  sp, #2
    jmp halt
```

All'uscita dalla funzione ***main()***, dopo aver salvato quanto restituito dalla funzione stessa, ci si introduce nel codice successivo al simbolo '**`halt`**', nel quale si chiama la funzione ***sys()*** (chiamata di sistema), per produrre la chiusura formale del processo. Ciò che si vede è comunque l'equivalente di '**`_exit (`*exit_status*`);`**'.[1]

[1] Va tenuto in considerazione che '**`exit_status`**' è un simbolo non raggiungibile dal codice C, perché dovrebbe essere esportato con un nome che inizi con il trattino basso.

Avvio del sistema e conclusione 1387
Interazione con il kernel 1387
Avvio e conclusione del sistema «normale» 1388
Sezione 1: programmi eseguibili o comandi interni di shell . 1391
os16: aaa(1) 1391
os16: bbb(1) 1392
os16: cat(1) 1392
os16: ccc(1) 1392
os16: chmod(1) 1392
os16: chown(1) 1392
os16: cp(1) 1393
os16: date(1) 1393
os16: ed(1) 1394
os16: kill(1) 1394
os16: ln(1) 1395
os16: login(1) 1395
os16: ls(1) 1396
os16: man(1) 1396
os16: mkdir(1) 1397
os16: more(1) 1397
os16: ps(1) 1398
os16: rm(1) 1399
os16: shell(1) 1399
os16: touch(1) 1400
os16: tty(1) 1400
Sezione 2: chiamate di sistema 1401
os16: _Exit(2) 1402
os16: _exit(2) 1402
os16: chdir(2) 1402
os16: chmod(2) 1403
os16: chown(2) 1404
os16: clock(2) 1405
os16: close(2) 1406
os16: dup(2) 1406
os16: dup2(2) 1407
os16: execve(2) 1407
os16: fchmod(2) 1408
os16: fchown(2) 1408
os16: fcntl(2) 1408
os16: fork(2) 1409
os16: fstat(2) 1410
os16: getcwd(2) 1410
os16: geteuid(2) 1410
os16: getuid(2) 1410
os16: getpgrp(2) 1411
os16: getpid(2) 1411
os16: getppid(2) 1412
os16: kill(2) 1412
os16: link(2) 1412
os16: lseek(2) 1413
os16: mkdir(2) 1414
os16: mknod(2) 1414
os16: mount(2) 1415
os16: open(2) 1416

os16: read(2) 1418
os16: rmdir(2) 1419
os16: seteuid(2) 1419
os16: setpgrp(2) 1420
os16: setuid(2) 1420
os16: signal(2) 1421
os16: sleep(2) 1422
os16: stat(2) 1422
os16: sys(2) 1424
os16: stime(2) 1425
os16: time(2) 1425
os16: umask(2) 1426
os16: umount(2) 1426
os16: unlink(2) 1426
os16: wait(2) 1427
os16: write(2) 1427
os16: z(2) 1428
os16: z_perror(2) 1429
os16: z_printf(2) 1429
os16: z_putchar(2) 1429
os16: z_puts(2) 1429
os16: z_vprintf(2) 1429

Sezione 3: funzioni di libreria 1431

os16: access(3) 1434
os16: abort(3) 1435
os16: abs(3) 1435
os16: atexit(3) 1435
os16: atoi(3) 1436
os16: atol(3) 1437
os16: basename(3) 1437
os16: bp(3) 1437
os16: clearerr(3) 1437
os16: closedir(3) 1438
os16: creat(3) 1438
os16: cs(3) 1438
os16: ctime(3) 1439
os16: dirname(3) 1440
os16: div(3) 1440
os16: ds(3) 1441
os16: endpwent(3) 1441
os16: errno(3) 1441
os16: es(3) 1445
os16: exec(3) 1445
os16: execl(3) 1446
os16: execle(3) 1446
os16: execlp(3) 1446
os16: execv(3) 1446
os16: execvp(3) 1446
os16: exit(3) 1446
os16: fclose(3) 1446
os16: feof(3) 1447
os16: ferror(3) 1447
os16: fflush(3) 1448
os16: fgetc(3) 1448
os16: fgetpos(3) 1448
os16: fgets(3) 1449
os16: fileno(3) 1450
os16: fopen(3) 1451

os16: fprintf(3) 1452
os16: fputc(3) 1452
os16: fputs(3) 1452
os16: fread(3) 1453
os16: free(3) 1453
os16: freopen(3) 1453
os16: fscanf(3) 1453
os16: fseek(3) 1453
os16: fseeko(3) 1454
os16: fsetpos(3) 1454
os16: ftell(3) 1454
os16: ftello(3) 1455
os16: fwrite(3) 1455
os16: getc(3) 1455
os16: getchar(3) 1455
os16: getenv(3) 1455
os16: getopt(3) 1456
os16: getpwent(3) 1459
os16: getpwnam(3) 1460
os16: getpwuid(3) 1461
os16: gets(3) 1461
os16: heap(3) 1461
os16: heap_clear(3) 1462
os16: heap_min(3) 1462
os16: input_line(3) 1462
os16: isatty(3) 1463
os16: labs(3) 1463
os16: ldiv(3) 1463
os16: major(3) 1463
os16: makedev(3) 1463
os16: malloc(3) 1464
os16: memccpy(3) 1465
os16: memchr(3) 1465
os16: memcmp(3) 1465
os16: memcpy(3) 1466
os16: memmove(3) 1466
os16: memset(3) 1466
os16: minor(3) 1467
os16: namep(3) 1467
os16: offsetof(3) 1468
os16: opendir(3) 1468
os16: perror(3) 1468
os16: printf(3) 1469
os16: process_info(3) 1471
os16: putc(3) 1472
os16: putchar(3) 1472
os16: putenv(3) 1472
os16: puts(3) 1473
os16: qsort(3) 1473
os16: rand(3) 1473
os16: readdir(3) 1474
os16: realloc(3) 1474
os16: rewind(3) 1475
os16: rewinddir(3) 1475
os16: scanf(3) 1475
os16: seg_d(3) 1479
os16: seg_i(3) 1479
os16: setbuf(3) 1479

os16: setenv(3) 1479
os16: setpwent(3) 1480
os16: setvbuf(3) 1480
os16: snprintf(3) 1480
os16: sp(3) 1480
os16: sprintf(3) 1480
os16: srand(3) 1480
os16: ss(3) 1481
os16: sscanf(3) 1481
os16: stdio(3) 1481
os16: strcat(3) 1482
os16: strchr(3) 1482
os16: strcmp(3) 1483
os16: strcoll(3) 1483
os16: strcpy(3) 1483
os16: strcspn(3) 1484
os16: strdup(3) 1484
os16: strerror(3) 1484
os16: strlen(3) 1484
os16: strncat(3) 1485
os16: strncmp(3) 1485
os16: strncpy(3) 1485
os16: strpbrk(3) 1485
os16: strrchr(3) 1485
os16: strspn(3) 1486
os16: strstr(3) 1486
os16: strtok(3) 1486
os16: strtol(3) 1488
os16: strtoul(3) 1489
os16: strxfrm(3) 1489
os16: ttyname(3) 1489
os16: unsetenv(3) 1490
os16: vfprintf(3) 1490
os16: vfscanf(3) 1490
os16: vprintf(3) 1490
os16: vscanf(3) 1491
os16: vsnprintf(3) 1492
os16: vsprintf(3) 1492
os16: vsscanf(3) 1492

Sezione 4: file speciali 1493

os16: console(4) 1493
os16: dsk(4) 1493
os16: kmem_file(4) 1494
os16: kmem_inode(4) 1494
os16: kmem_mmp(4) 1494
os16: kmem_ps(4) 1495
os16: kmem_sb(4) 1495
os16: mem(4) 1496
os16: null(4) 1496
os16: port(4) 1496
os16: tty(4) 1496
os16: zero(4) 1497

Sezione 5: formato dei file e convenzioni 1499

os16: inittab(5) 1499
os16: issue(5) 1499
os16: passwd(5) 1499

Sezione 7: varie 1501

os16: environ(7) 1501
os16: undocumented(7) 1501

Sezione 8: comandi per l’amministrazione del sistema 1503

os16: getty(8) 1503
os16: init(8) 1503
os16: MAKEDEV(8) 1504
os16: mount(8) 1504
os16: umount(8) 1505

Sezione 9: kernel 1507

os16: devices(9) 1509
os16: diag(9) 1514
os16: fs(9) 1514
os16: ibm_i86(9) 1556
os16: k_libc(9) 1560
os16: main(9) 1560
os16: memory(9) 1560
os16: proc(9) 1561
os16: tty(9) 1585

Interazione con il kernel 1387
Avvio e conclusione del sistema «normale» 1388

os16 ha due modalità di funzionamento: si può interagire direttamente con il kernel, oppure si può avviare il processo '**init**' e procedere con l'organizzazione consueta di un sistema Unix tradizionale. La modalità di colloquio diretto con il kernel è servita per consentire lo sviluppo di os16 e potrebbe essere utile per motivi di studio. Va osservato che durante l'interazione diretta con il kernel si dispone di una sola console, mentre quando si avvia '**init**' si possono avere delle console virtuali in base alla configurazione.

Interazione con il kernel

L'avvio di os16 passa, in ogni caso, per una prima fase di colloquio con il kernel. Si ottiene un menù e si possono premere semplicemente dei tasti, secondo l'elenco previsto, per ottenere delle azioni molto semplici. In questa fase il disco da cui risulta avviato il kernel è già innestato ed è prevista la possibilità di avviare tre programmi: '`/bin/aaa`', '`/bin/bbb`' e '`/bin/ccc`'. In tal modo, si ha la possibilità di avviare qualcosa, a titolo diagnostico, prima dello stesso '**init**' ('`/bin/init`'.

Figura u151.1. Aspetto di os16 in funzione, con il menù in evidenza.

```
os16 build 20AA.MM.GG HH:MM:SS ram 639 Kibyte
.------------------------------------------------------------.
| [h]      show this menu                                    |
| [p]      process status and memory map                     |
| [1]..[9] kill process  1 to 9                              |
| [A]..[F] kill process 10 to 15                             |
| [l]      send SIGCHLD to process 1                         |
| [a]..[c] run programs '/bin/aaa' to '/bin/ccc' in parallel |
| [f]      system file status                                |
| [n], [N] list of active inodes                             |
| [z]      print root file system zone map (read left to right) |
| [m], [M] mount/umount '/dev/dsk1' at '/usr/'               |
| [x]      exit interaction with kernel and start '/bin/init' |
| [q]      quit kernel                                       |
'------------------------------------------------------------'
```

Le funzioni principali disponibili in questa modalità diagnostica sono riassunte nella tabella successiva:

Tasto	Risultato
[*h*]	Mostra il menù di funzioni disponibili.
[*1*]	Invia il segnale '**SIGKILL**' al processo numero uno.
[*2*]...[*9*]	Invia il segnale '**SIGTERM**' al processo con il numero corrispondente.
[*A*]...[*F*]	Invia il segnale '**SIGTERM**' al processo con il numero da 10 a 15.
[*a*], [*b*], [*c*]	Avvia il programma '`/bin/aaa`', '`/bin/bbb`' o '`/bin/ccc`'.
[*f*]	Mostra l'elenco dei file aperti nel sistema.
[*m*], [*M*]	Innesta o stacca il secondo dischetto dalla directory '`/usr/`'.
[*n*], [*N*]	Mostra l'elenco degli inode aperti: l'elenco è composto da due parti.
[*l*]	Invia il segnale '**SIGCHLD**' al processo numero uno.
[*p*]	Mostra la situazione dei processi e altre informazioni.
[*x*]	Termina il ciclo e successivamente si passa all'avvio di '`/bin/init`'.
[*q*]	Ferma il sistema.

Figura u151.3. Aspetto di os16 in funzione mentre visualizza anche la tabella dei processi avviati (tasto [*p*]).

```
ababaaababaaababaabbaaababaaababap
pp  p pg
id id rp  tty  uid euid suid usage s iaddr isiz daddr dsiz sp   name
 0  0  0 0000    0    0    0 00.35 R 10500 eb7c 00500 0000 ffc8 os16 kernel
 0  1  0 0000    0    0    0 00.33 r 2f100 0600 2f700 aa00 a8e8 /bin/ccc
 0  2  0 0000    0    0    0 00.01 r 1f100 0600 84300 aa00 a8e8 /bin/ccc
 2  3  0 0000   10   10   10 00.01 r 1f100 0600 44b00 aa00 a8e8 /bin/ccc
 0  4  0 0000    0    0    0 00.17 r 21600 0600 3a100 aa00 a8e8 /bin/ccc
 4  5  0 0000   10   10   10 00.02 r 21c00 2400 6f100 a900 a86c /bin/aaa
 4  6  0 0000   11   11   11 00.02 s 24000 2500 59d00 a900 a8b6 /bin/bbb
 0  7  0 0000    0    0    0 00.13 r 26500 0600 64600 aa00 a8e8 /bin/ccc
 7  8  0 0000   10   10   10 00.02 r 26b00 2400 8ee00 a900 981e /bin/aaa
 7  9  0 0000   11   11   11 00.02 s 2bf00 2500 79a00 a900 a8b6 /bin/bbb
CS=1050 DS=0050 SS=0050 ES=0050 BP=ffe4 SP=ffe4 heap_min=878c etext=eb7c edata=1
b3c ebss=4c34 ksp=ffc8 clock=0000084b, time elapsed=00:01:57
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffe00000003fffffffffffffffffffffff
fffe000000000001fffffffff0007fffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff800
00000000000000000000000000000000000000007fffffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffbfffffffffffffffffffffffffffffffffffffffffffe0000000000000000000000f


abbaaaaabaaaabaaaabaaabbaabbaabababbaabbbbbbbbbbbbbbb
```

Premendo [*x*], il ciclo termina e il kernel avvia ‘`/bin/init`’, ma prima di farlo occorre che non ci siano altri processi in funzione, perché ‘**`init`**’ deve assumere il ruolo di processo 1, ovvero il primo dopo il kernel.

Figura u151.4. Aspetto di os16 in funzione con il menù in evidenza, dopo aver premuto il tasto [*x*] per avviare ‘**`init`**’.

```
os16 build 20AA.MM.GG HH:MM:SS ram 639 Kibyte
.-----------------------------------------------------------------.
| [h]      show this menu                                         |
| [p]      process status and memory map                          |
| [1]..[9] kill process  1 to 9                                   |
| [A]..[F] kill process 10 to 15                                  |
| [l]      send SIGCHLD to process 1                              |
| [a]..[c] run programs '/bin/aaa' to '/bin/ccc' in parallel      |
| [f]      system file status                                     |
| [n], [N] list of active inodes                                  |
| [z]      print root file system zone map (read left to right)   |
| [m], [M] mount/umount '/dev/dsk1' at '/usr/'                    |
| [x]      exit interaction with kernel and start '/bin/init'     |
| [q]      quit kernel                                            |
'-----------------------------------------------------------------'
init
os16: a basic os. [Ctrl q], [Ctrl r], [Ctrl s], [Ctrl t] to change
console.
This is terminal /dev/console0
Log in as "root" or "user" with password "ciao" :-)
login:
```

Avvio e conclusione del sistema «normale»

Se non si intende operare direttamente con il kernel, come descritto nella sezione precedente, con la pressione del tasto [*x*] si avvia ‘**`init`**’.

Il programma ‘**`init`**’ legge il file ‘`/etc/inittab`’ e sulla base del suo contenuto, avvia uno o più processi ‘**`getty`**’, per la gestione dei vari terminali disponibili (si tratta comunque soltanto di console virtuali).

Il programma ‘**`getty`**’ apre il terminale che gli viene indicato come opzione della chiamata (da ‘**`init`**’ che lo determina in base al contenuto di ‘`/etc/inittab`’), facendo in modo che sia associato al descrittore zero (standard input). Quindi, dopo aver visualizzato il contenuto del file ‘`/etc/issue`’, mostra un proprio messaggio e avvia il programma ‘**`login`**’.

Il programma ‘**`login`**’ prende il posto di ‘**`getty`**’ che così scompare dall’elenco dei processi. ‘**`login`**’ procede chiedendo all’utente di identificarsi, utilizzando il file ‘`/etc/passwd`’ per verificare le credenziali di accesso. Se l’identificazione ha successo, viene avviata la shell definita nel file ‘`/etc/passwd`’ per l’utente, in modo da sostituirsi al programma ‘**`login`**’, il quale scompare a sua volta dall’elenco dei processi.

Attraverso la shell è possibile interagire con il sistema operativo, secondo la modalità «normale», nei limiti delle possibilità di os16. Quando la shell termina di funzionare, ‘**`init`**’ riavvia ‘**`getty`**’.

Per cambiare console virtuale si possono usare le combinazioni [*Ctrl q*], [*Ctrl r*], [*Ctrl s*] e [*Ctrl t*], ma bisogna considerare che dipende dalla configurazione del file ‘`/etc/inittab`’ se effettivamente vengono attivate tutte queste console.

Per concludere l’attività del sistema, basta concludere il funzionamento delle varie sessioni di lavoro (la shell finisce di funzionare con il comando interno ‘**`exit`**’) e non c’è bisogno di altro; pertanto, non è previsto l’uso di comandi come ‘**`halt`**’ o ‘**`shutdown`**’ e, d’altro canto, le operazioni di scrittura nel file system sono sincrone, in modo tale da non richiedere accorgimenti particolari per la chiusura delle attività.

Riquadro u151.5. Coerenza del file system.

os16 è destinato a essere usato attraverso un emulatore, dove gli unici due dischi previsti (dischetti da 1440 Kibyte) sono semplicemente dei file. Pertanto, la manutenzione del file system, avviene all’esterno di os16, di norma con un sistema GNU/Linux. Tuttavia, occorre tenere presente che quando si usa il sistema os16, tramite l’emulatore, i file-immagine dei dischi di os16 non devono essere innestati nel sistema operativo ospitante, perché altrimenti le operazioni di scrittura eseguite da os16 potrebbero essere annullate, anche solo parzialmente, dalla gestione del sistema ospitante, rendendo il file system incoerente.

os16: aaa(1) 1391
os16: bbb(1) 1392
os16: cat(1) 1392
os16: ccc(1) 1392
os16: chmod(1) 1392
os16: chown(1) 1392
os16: cp(1) 1393
os16: date(1) 1393
os16: ed(1) 1394
os16: kill(1) 1394
os16: ln(1) 1395
os16: login(1) 1395
os16: ls(1) 1396
os16: man(1) 1396
os16: mkdir(1) 1397
os16: more(1) 1397
os16: ps(1) 1398
os16: rm(1) 1399
os16: shell(1) 1399
os16: touch(1) 1400
os16: tty(1) 1400

`aaa` 1391 `cat` 1392 `chmod` 1392 `chown` 1392 `cp` 1393 `date` 1393 `ed` 1394 `kill` 1394 `ln` 1395 `login` 1395 `ls` 1396 `man` 1396 `mkdir` 1397 `more` 1397 `ps` 1398 `rm` 1399 `shell` 1399 `touch` 1400 `tty` 1400

os16: aaa(1)

NOME

‘**aaa**’, ‘**bbb**’, ‘**ccc**’ - programmi elementari avviabili direttamente dal kernel

SINTASSI

```
aaa
```

```
bbb
```

```
ccc
```

DESCRIZIONE

‘**aaa**’ e ‘**bbb**’ si limitano a visualizzare una lettera, rispettivamente «a» e «b», attraverso lo standard output, a intervalli regolari. Precisamente, ‘**aaa**’ lo fa a intervalli di un secondo, mentre ‘**bbb**’ a intervalli di due secondi. Il lavoro di ‘**aaa**’ e di ‘**bbb**’ si conclude dopo l’emissione, rispettivamente, di 60 e 30 caratteri, pertanto nel giro di un minuto di tempo si esaurisce il loro compito.

Il programma ‘**ccc**’ è diverso, ma nasce per lo stesso scopo: controllare la gestione dei processi di os16. Questo programma si limita ad avviare, ‘**aaa**’ e ‘**bbb**’, come propri processi-figli, rimanendo in funzione, senza fare nulla. Pertanto, se si usa ‘**ccc**’, il suo processo deve essere eliminato in modo esplicito, perché da solo non si concluderebbe mai.

Questi programmi sono indicati soprattutto per l’uso di os16 nella modalità interattiva che precede il funzionamento normale del sistema operativo, per la verifica della gestione dei processi.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/aaa.c`’ [i162.1.2]

‘`applic/bbb.c`’ [i162.1.3]

‘`applic/ccc.c`’ [i162.1.5]

os16: bbb(1)

«

Vedere *aaa(1)* [u0.1].

os16: cat(1)

«

NOME

‘**`cat`**’ - emissione del contenuto di uno o più file attraverso lo standard output

SINTASSI

```
cat [file]...
```

DESCRIZIONE

‘**`cat`**’ legge il contenuto dei file indicati come argomento e li emette attraverso lo standard output, concatenati assieme in un unico flusso.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/cat.c`’ [i162.1.4]

VEDERE ANCHE

more(1) [u0.16], *ed(1)* [u0.9].

os16: ccc(1)

«

Vedere *aaa(1)* [u0.1].

os16: chmod(1)

«

NOME

‘**`chmod`**’ - cambiamento della modalità dei permessi dei file

SINTASSI

```
chmod mod_ottale file...
```

DESCRIZIONE

‘**`chmod`**’ cambia la modalità dei permessi associati ai file indicati, in base al numero ottale indicato come primo argomento.

NOTE

Questa versione di ‘**`chmod`**’ non permette di indicare la modalità dei permessi in forma simbolica.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/chmod.c`’ [i162.1.6]

VEDERE ANCHE

chown(1) [u0.6].

os16: chown(1)

«

NOME

‘**`chown`**’ - cambiamento del proprietario di un file

SINTASSI

```
chown nome_utente file...
```

```
chown uid file...
```


DESCRIZIONE

‘**`chown`**’ cambia l’utente proprietario dei file indicati. Il nuovo proprietario da attribuire può essere indicato per nome o per numero.

NOTE

os16 non gestisce i gruppi, pertanto si può intervenire soltanto sull’utente proprietario dei file.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/chown.c`’ [i162.1.7]

VEDERE ANCHE

chmod(1) [u0.5].

os16: cp(1)

«

NOME

‘**`cp`**’ - copia dei file

SINTASSI

```
cp file_orig file_nuovo...
```

```
cp file... directory_dest...
```

DESCRIZIONE

‘**`cp`**’ copia uno o più file. Se l’ultimo argomento è costituito da una directory esistente, la copia produce dei file con lo stesso nome degli originali, all’interno della directory; se l’ultimo argomento non è una directory già esistente, ci può essere un solo file da copiare, intendendo che si voglia creare una copia con quel nome specificato.

DIFETTI

Non è possibile copiare oggetti diversi dai file puri e semplici; quindi, la copia ricorsiva di una directory non è ammissibile.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/cp.c`’ [i162.1.8]

VEDERE ANCHE

touch(1) [u0.20], *mkdir(1)* [u0.15].

os16: date(1)

«

NOME

‘**`date`**’ - visualizzazione o impostazione della data e dell’ora di sistema

SINTASSI

```
date [MMGGhhmm[[SS]AA]]
```

DESCRIZIONE

Se si utilizza il programma ‘**`date`**’ senza argomenti, si ottiene la visualizzazione della data e dell’ora attuale del sistema operativo. Se si indica una sequenza numerica come argomento, si intende invece impostare la data e l’ora del sistema. In tal caso va indicato un numero preciso di cifre, che può essere di otto, dieci o dodici. Se si immettono otto cifre, si sta specificando il mese, il giorno, l’ora e i minuti dell’anno attuale; se si indicano dieci cifre, le ultime due rappresentano l’anno del secolo attuale; se si immettono dodici cifre, l’anno è indicato per esteso nelle ultime quattro cifre.

ESEMPI

```
# date 123123592012 [Invio]
```

Imposta la data di sistema al giorno 31 dicembre 2012, alle ore 23:59.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/date.c`’ [i162.1.10]

VEDERE ANCHE

time(2) [u0.39], *stime(2)* [u0.39].

os16: ed(1)

NOME

‘**ed**’ - creazione e modifica di file di testo

SINTASSI

```
ed
```

DESCRIZIONE

‘**ed**’ è un programma di creazione e modifica di file di testo, che consente di operare su una riga alla volta.

‘**ed**’ opera in due modalità di funzionamento: comando e inserimento. All’avvio, ‘**ed**’ si trova in modalità di comando, durante la quale ciò che si inserisce attraverso lo standard input viene interpretato come un comando da eseguire. Per esempio, il comando ‘**1i**’ richiede di passare alla modalità di inserimento, immettendo delle righe a partire dalla prima posizione, spostando quelle presenti in basso. Quando ci si trova in modalità di inserimento, per poter passare alla modalità di comando si introduce un punto isolato, all’inizio di una nuova riga.

Per il momento, in questa pagina di manuale, si omette la descrizione completa dell’utilizzo di ‘**ed**’.

DIFETTI

La digitazione da tastiera viene interpretata da ‘**ed**’ in modo letterale. Pertanto, anche la cancellazione, [*Backspace*], benché visivamente faccia indietreggiare il cursore, in realtà introduce il codice <*DEL*>. In fase di inserimento ciò comporta la scrittura di tale codice; in modalità di comando, ciò rende errato l’inserimento.

Il file che si intende elaborare con ‘**ed**’ viene caricato o creato completamente nella memoria centrale. Dal momento che os16 consente a ogni processo di gestire una quantità molto limitata di memoria, si può lavorare soltanto con file di dimensioni estremamente ridotte.

AUTORI

Questa edizione di ‘**ed**’ è stata scritta originariamente da David I. Bell, per ‘**sash**’ (una shell che include varie funzionalità, da compilare in modo statico). Successivamente, il codice è stato estrapolato da ‘**sash**’ e reso indipendente, per gli scopi del sistema operativo ELKS (una versione a 16 bit di Linux). Dalla versione estratta per ELKS è stata ottenuta quella di os16, con una serie di modifiche apportate da Daniele Giacomini, tra cui risulta particolarmente evidente il cambiamento dello stile di impaginazione del codice.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/ed.c`’ [i162.1.11]

VEDERE ANCHE

shell(1) [u0.19].

os16: kill(1)

NOME

‘**kill**’ - invio di un segnale ai processi

SINTASSI

```
kill -s nome_segnale pid...
```



```
kill -l
```

DESCRIZIONE

Il programma ‘**kill**’ consente di inviare un segnale, indicato per nome, a uno o più processi, specificati per numero.

OPZIONI

Opzione	Descrizione
`-l`	Mostra l’elenco dei nomi dei segnali disponibili.
`-s` ***nome_segnale***	Specifica il nome del segnale da inviare ai processi.

NOTE

Non è possibile indicare il segnale per numero, perché lo standard definisce i nomi di un insieme di segnali necessari, ma non stabilisce il numero, il quale può essere attribuito liberamente in fase realizzativa.

DIFETTI

os16 non consente ai processi di attribuire azioni alternative ai segnali; pertanto, si possono ottenere solo quelle predefinite. Tutto quello che si può fare è, eventualmente, bloccare i segnali, esclusi però quelli che non sono mascherabili per loro natura.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/kill.c`’ [i162.1.14]

VEDERE ANCHE

kill(2) [u0.22], *signal(2)* [u0.34].

os16: ln(1)

NOME

‘**ln**’ - collegamento dei file

SINTASSI

```
ln file_orig file_nuovo...
```

```
ln file... directory_dest...
```

DESCRIZIONE

‘**ln**’ crea il collegamento fisico di uno o più file. Se l’ultimo argomento è costituito da una directory esistente, si producono collegamenti con gli stessi nomi degli originali, all’interno della directory; se l’ultimo argomento non è una directory già esistente, ci può essere un solo file da collegare, intendendo che si voglia creare un collegamento con quel nome specificato.

Essendo disponibile soltanto la creazione di collegamenti fisici, questi collegamenti possono essere collocati soltanto all’interno del file system di quelli originali, senza contare eventuali innesti ulteriori.

DIFETTI

Non è possibile creare dei collegamenti simbolici, perché os16 non sa gestirli.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/ln.c`’ [i162.1.15]

VEDERE ANCHE

cp(1) [u0.7], *link(2)* [u0.23].

os16: login(1)

NOME

‘**login**’ - inizio di una sessione presso un terminale

SINTASSI

```
login
```

DESCRIZIONE

‘**login**’ richiede l’inserimento di un nominativo-utente e di una parola d’ordine. Questa coppia viene verificata consultando il file ‘/etc/passwd’ e se coincide: vengono cambiati i permessi e la proprietà del file di dispositivo del terminale di controllo; viene cambiata la directory corrente in modo da farla coincidere con la directory personale dell’utente; viene avviata la shell, indicata sempre nel file ‘/etc/passwd’ per quel tale utente, con i privilegi di questo. La shell, avviata così, va a rimpiazzare il processo di ‘**login**’.

Il programma ‘**login**’ è fatto per essere avviato da ‘**getty**’, non avendo altri utilizzi pratici.

FILE

File	Descrizione
‘/etc/passwd’	Contiene l’elenco degli utenti, con l’associazione al numero UID, alla parola d’ordine necessaria per accedere, alla shell dell’utente. Le altre informazioni eventuali contenute nel file, non sono usate da ‘**login**’.

FILE SORGENTI

‘applic/crt0.s’ [i162.1.9]

‘applic/login.c’ [i162.1.16]

VEDERE ANCHE

getty(8) [u0.1], *console(4)* [u0.1].

os16: ls(1)

NOME

‘**ls**’ - elenco del contenuto delle directory

SINTASSI

```
ls [opzioni] [file]...
```

DESCRIZIONE

‘**ls**’ elenca i file e le directory indicati come argomenti della chiamata. Se non vengono indicati file o directory da visualizzare, si ottiene l’elenco del contenuto della directory corrente; inoltre, questa realizzazione particolare di ‘**ls**’, se si indica come argomento solo una directory, ne mostra il contenuto, altrimenti, se gli argomenti sono più di uno, mostra solo i nomi richiesti, eventualmente con le rispettive caratteristiche se è stata usata l’opzione ‘**-l**’.

OPZIONI

Opzione	Descrizione
-a	Quando si richiede di mostrare il contenuto di una directory (quella corrente o quella specificata espressamente come primo e unico argomento), con questa opzione si ottiene la visualizzazione anche dei nomi che iniziano con un punto, inclusi ‘.’ e ‘..’.
-l	Con questa opzione si ottiene la visualizzazione di più informazioni sui file e sulle directory elencati.

NOTE

Dal momento che os16 non considera i gruppi, quando si usa l’opzione ‘**-l**’, il nome del gruppo a cui appartiene un file o una directory, non viene visualizzato.

FILE SORGENTI

‘applic/crt0.s’ [i162.1.9]

‘applic/ls.c’ [i162.1.17]

os16: man(1)

NOME

‘**man**’ - visualizzazione delle pagine di manuale

SINTASSI

```
man [sezione] pagina
```

DESCRIZIONE

‘**man**’ visualizza la pagina di manuale indicata come argomento, consentendone lo scorrimento in avanti. La «pagina» viene cercata tra le sezioni, a partire dalla prima. In caso di omonimie tra sezioni differenti, si può specificare il numero della sezione prima del nome della pagina.

Le pagine di manuale di os16 sono semplicemente dei file di testo, collocati nella directory ‘/usr/share/man/’, con nomi del tipo ‘***pagina*.*n***’, dove ***n*** è il numero della sezione.

FILE SORGENTI

‘applic/crt0.s’ [i162.1.9]

‘applic/man.c’ [i162.1.18]

VEDERE ANCHE

cat(1) [u0.3].

os16: mkdir(1)

NOME

‘**mkdir**’ - creazione di directory

SINTASSI

```
mkdir [-p] [-m mod_ottale] [directory]...
```

DESCRIZIONE

‘**mkdir**’ crea una o più directory, corrispondenti ai nomi che costituiscono gli argomenti.

OPZIONI

Opzione	Descrizione
-p	Se la directory che si vuole creare, può richiedere prima la creazione di altre directory, con questa opzione (*parents*) si generano tutte le directory genitrici necessarie, purché quei nomi non siano già usati per dei file.
-m *mod_ottale*	Quando si crea una directory, senza specificare questa opzione, si ottengono i permessi 0777_8 meno quanto contenuto nella maschera di creazione dei file e delle directory. Con l’opzione ‘**-m**’ si vanno invece a specificare i permessi desiderati in modo esplicito.

FILE SORGENTI

‘applic/crt0.s’ [i162.1.9]

‘applic/mkdir.c’ [i162.1.19]

VEDERE ANCHE

mkdir(2) [u0.25], *rmdir(2)* [u0.30].

os16: more(1)

NOME

‘**more**’ - visualizzazione di file sullo schermo, permettendo il controllo dello scorrimento dei dati, ma in un solo verso

SINTASSI

```
more file...
```

DESCRIZIONE

'**more**' visualizza i file indicati come argomenti della chiamata, sospendendo lo scorrimento del testo dopo un certo numero di righe, consentendo all'utente di decidere come proseguire.

COMANDI

Quando '**more**' sospende lo scorrimento del testo, è possibile introdurre un comando, composto da un solo carattere, tenendo conto che ciò che non è previsto fa comunque proseguire lo scorrimento:

Comando	Descrizione
[*n*]	si richiede di saltare al file successivo, ammesso che ce ne sia un altro;
[*q*]	si richiede di interrompere lo scorrimento e di concludere il funzionamento del programma;
[*Spazio*]	si richiede di proseguire nella visualizzazione progressiva dei file.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`applic/more.c`' [i162.1.20]

VEDERE ANCHE

cat(1) [u0.3].

os16: ps(1)

NOME

'**ps**' - visualizzazione dello stato dei processi elaborativi

SINTASSI

```
ps
```

DESCRIZIONE

'**ps**' visualizza l'elenco dei processi, con le informazioni disponibili sul loro stato. L'elenco è provvisto di un'intestazione, come si vede nell'esempio seguente:

```
pp  p pg
id id rp  tty  uid euid suid usage s iaddr isiz daddr dsiz sp   name
 0  0  0 0000    0    0    0 00.31 r 10500 eff8 00500 0000 ffca os16 kernel
 0  1  0 0000    0    0    0 00.15 s 2f500 3200 32700 3300 2b8c /bin/init
 1  2  2 0500 1001 1001 1001 00.15 s 25700 3900 29000 3400 2f8c /bin/shell
 1  3  3 0501    0    0    0 00.07 s 22600 3100 38e00 3400 2556 /bin/login
 2  4  2 0500 1001 1001 1001 00.03 R 1f500 2b00 3c200 3400 3172 /bin/ps
```

La prima colonna, con la sigla «ppid», ovvero *parent pid*, riporta il numero del processo genitore; la seconda, con la sigla «pid», *process id*, indica il numero del processo preso in considerazione; la terza, con la sigla «pgrp», *process group*, indica il gruppo a cui appartiene il processo; la quarta colonna, «tty», indica il terminale associato, ammesso che ci sia, come numero di dispositivo, ma in base sedici. Le colonne «uid», «euid» e «suid», riguardano l'identità dell'utente, per conto del quale sono in funzione i processi, rappresentando, nell'ordine, l'identità reale (*real user id*), quella efficace (*effective user id*) e quella salvata precedentemente (*saved user id*).

La colonna «usage» indica il tempo di utilizzo della CPU; la colonna «s» indica lo stato del processo, il cui significato può essere interpretato con l'aiuto della tabella successiva:

Lettera	Significato	Descrizione
R	*running*	in corso di esecuzione
r	*ready*	pronto per essere messo in esecuzione
s	*sleeping*	in attesa
z	*zombie*	terminato e non più in memoria, per il quale si attende di passare lo stato di uscita al processo genitore.

Le colonne «iaddr» e «isiz» indicano l'indirizzo iniziale e l'estensione dell'area codice del processo, in memoria; le colonne «daddr» e «dsiz» indicano l'indirizzo iniziale e l'estensione dell'area dati del processo, in memoria. La colonna «sp» indica il valore dell'indice della pila dei dati (*stack pointer*).

L'ultima colonna indica il nome del programma, assieme al suo percorso, con il quale il processo è stato avviato.

NOTE

L'elenco dei processi include anche il kernel, il quale occupa correttamente la prima posizione (processo zero).

FILE

'**ps**' trae le informazioni sullo stato dei processi da un file di dispositivo speciale: '`/dev/kmem_ps`'.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`applic/ps.c`' [i162.1.22]

os16: rm(1)

NOME

'**rm**' - cancellazione di file

SINTASSI

```
rm file...
```

DESCRIZIONE

'**rm**' consente di cancellare i file indicati come argomento.

DIFETTI

Non è possibile eseguire la cancellazione ricorsiva di una directory.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`applic/rm.c`' [i162.1.23]

VEDERE ANCHE

unlink(2) [u0.42].

os16: shell(1)

NOME

'**shell**' - interprete dei comandi

SINTASSI

```
shell
```

DESCRIZIONE

'**shell**' è l'interprete dei comandi di os16. Di norma viene avviato da '**login**', in base alla configurazione contenuta nel file '`/etc/passwd`'.

'**shell**' interpreta i comandi inseriti; se si tratta di un comando interno lo esegue direttamente, altrimenti cerca e avvia un programma con il nome corrispondente, rimanendo in attesa fino alla conclusione del processo relativo, per riprendere poi il controllo.

DIFETTI

L'interpretazione della riga di comando è letterale, pertanto non c'è alcuna espansione di caratteri speciali, variabili di ambiente o altro; inoltre, non è possibile eseguire script.

A volte, quando un processo avviato da '**shell**' termina di funzionare, il processo di '**shell**' non viene risvegliato correttamente, rendendo inutilizzabile il terminale. Per ovviare all'inconveniente, si può premere la combinazione [*Ctrl c*], con la quale viene inviato il segnale '**SIGINT**' a tutti i processi del gruppo associato al terminale.

Anche il fatto che un segnale generato con una combinazione di tasti si trasmetta a tutti i processi del gruppo associato al terminale è un'anomalia, tuttavia fa parte delle particolarità dovute alla semplificazione di os16.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/shell.c`’ [i162.1.24]

VEDERE ANCHE

input_line(3) [u0.60].

os16: touch(1)

NOME

‘**`touch`**’ - creazione di un file vuoto oppure aggiornamento della data di modifica

SINTASSI

```
touch file…
```

DESCRIZIONE

‘**`touch`**’ crea dei file vuoti, se quelli indicati come argomento non sono esistenti; altrimenti, aggiorna le date di accesso e di modifica, sulla base dello stato dell’orologio di sistema.

DIFETTI

Non è possibile attribuire una data arbitraria; inoltre, a causa della limitazione del tipo di file system utilizzato, non è possibile distinguere tra date di accesso e modifica dei file.

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/touch.c`’ [i162.1.25]

os16: tty(1)

NOME

‘**`tty`**’ - nome del file di dispositivo del terminale associato allo standard input

SINTASSI

```
tty
```

DESCRIZIONE

Il programma ‘**`tty`**’ individua il dispositivo del terminale associato allo standard input e lo traduce in un percorso che descrive il file di dispositivo corrispondente (ovvero il file di dispositivo che dovrebbe corrispondergli)

FILE SORGENTI

‘`applic/crt0.s`’ [i162.1.9]

‘`applic/tty.c`’ [i162.1.26]

Sezione 2: chiamate di sistema

os16: _Exit(2) 1402
os16: _exit(2) 1402
os16: chdir(2) 1402
os16: chmod(2) 1403
os16: chown(2) 1404
os16: clock(2) 1405
os16: close(2) 1406
os16: dup(2) 1406
os16: dup2(2) 1407
os16: execve(2) 1407
os16: fchmod(2) 1408
os16: fchown(2) 1408
os16: fcntl(2) 1408
os16: fork(2) 1409
os16: fstat(2) 1410
os16: getcwd(2) 1410
os16: geteuid(2) 1410
os16: getuid(2) 1410
os16: getpgrp(2) 1411
os16: getpid(2) 1411
os16: getppid(2) 1412
os16: kill(2) 1412
os16: link(2) 1412
os16: lseek(2) 1413
os16: mkdir(2) 1414
os16: mknod(2) 1414
os16: mount(2) 1415
os16: open(2) 1416
os16: read(2) 1418
os16: rmdir(2) 1419
os16: seteuid(2) 1419
os16: setpgrp(2) 1420
os16: setuid(2) 1420
os16: signal(2) 1421
os16: sleep(2) 1422
os16: stat(2) 1422
os16: sys(2) 1424
os16: stime(2) 1425
os16: time(2) 1425
os16: umask(2) 1426
os16: umount(2) 1426
os16: unlink(2) 1426
os16: wait(2) 1427
os16: write(2) 1427
os16: z(2) 1428
os16: z_perror(2) 1429
os16: z_printf(2) 1429
os16: z_putchar(2) 1429
os16: z_puts(2) 1429

os16: z_vprintf(2) 1429

chdir() 1402 chmod() 1403 chown() 1404 clock() 1405
close() 1406 dup() 1406 dup2() 1406 execve() 1407
fchmod() 1403 fchown() 1404 fcntl() 1408 fork() 1409
fstat() 1422 getcwd() 1410 geteuid() 1410
getpgrp() 1411 getpid() 1411 getppid() 1411
getuid() 1410 kill() 1412 link() 1412 lseek() 1413
mkdir() 1414 mknod() 1414 mount() 1415 open() 1416
read() 1418 rmdir() 1419 seteuid() 1420 setpgrp()
1420 setuid() 1420 signal() 1421 sleep() 1422 stat()
1422 stime() 1425 sys() 1424 time() 1425 umask() 1426
umount() 1415 unlink() 1426 wait() 1427 write() 1427
z_perror() 1428 z_printf() 1428 z_putchar() 1428
z_puts() 1428 z_vprintf() 1428 _exit() 1402
_Exit() 1402

os16: _Exit(2)

Vedere *_exit(2)* [u0.2].

os16: _exit(2)

NOME

'**_exit**', '**_Exit**' - conclusione del processo chiamante

SINTASSI

```
#include <unistd.h>
void _exit (int status);
```

```
#include <stdlib.h>
void _Exit (int status);
```

DESCRIZIONE

Le funzioni ***_exit()*** e ***_Exit()*** sono equivalenti e servono per concludere il processo chiamante, con un valore pari a quello indicato come argomento (***status***), purché inferiore o uguale 255 (FF_{16}).

La conclusione del processo implica anche la chiusura dei suoi file aperti, e l'affidamento di eventuali processi figli al processo numero uno ('**init**'); inoltre, si ottiene l'invio di un segnale SIGCHLD al processo genitore di quello che viene concluso.

VALORE RESTITUITO

La funzione non può restituire alcunché, dal momento che la sua esecuzione comporta la morte del processo.

FILE SORGENTI

'lib/unistd.h' [u0.17]

'lib/unistd/_exit.c' [i161.17.1]

'lib/stdlib.h' [u0.10]

'lib/stdlib/_Exit.c' [i161.10.1]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_exit.c' [i160.9.22]

VEDERE ANCHE

execve(2) [u0.10], *fork(2)* [u0.14], *kill(2)* [u0.22], *wait(2)* [u0.43], *atexit(3)* [u0.4], *exit(3)* [u0.4].

os16: chdir(2)

NOME

'**chdir**' - modifica della directory corrente

SINTASSI

```
#include <unistd.h>
int chdir (const char *path);
```


DESCRIZIONE

La funzione ***chdir()*** cambia la directory corrente, in modo che quella nuova corrisponda al percorso annotato nella stringa ***path***.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EINVAL	Argomento non valido.
EACCES	Accesso negato.
ENOTDIR	Uno dei componenti del percorso non è una directory.
ENOENT	Uno dei componenti del percorso non esiste.

FILE SORGENTI

'lib/unistd.h' [u0.17]

'lib/unistd/chdir.c' [i161.17.3]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/fs/path_chdir.c' [i160.4.32]

VEDERE ANCHE

rmdir(2) [u0.30], *access(3)* [u0.1].

os16: chmod(2)

NOME

'**chmod**', '**fchmod**' - cambiamento della modalità dei permessi di un file

SINTASSI

```
#include <sys/stat.h>
int chmod  (const char *path, mode_t mode);
int fchmod (int fdn, mode_t mode);
```

DESCRIZIONE

Le funzioni ***chmod()*** e ***fchmod()*** consentono di modificare la modalità dei permessi di accesso di un file. La funzione ***chmod()*** individua il file su cui intervenire attraverso un percorso, ovvero la stringa ***path***; la funzione ***fchmod()***, invece, richiede l'indicazione del numero di un descrittore di file, già aperto. In entrambi i casi, l'ultimo argomento serve a specificare la nuova modalità dei permessi.

Tradizionalmente, i permessi si scrivono attraverso un numero in base otto; in alternativa, si possono usare convenientemente della macro-variabili, dichiarate nel file 'sys/stat.h', combinate assieme con l'operatore binario OR.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	00700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	00400_8	Lettura per l'utente proprietario.
`S_IWUSR`	00200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	00100_8	Esecuzione per l'utente proprietario.
`S_IRWXG`	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	00040_8	Lettura per il gruppo.
`S_IWGRP`	00020_8	Scrittura per il gruppo.
`S_IXGRP`	00010_8	Esecuzione per il gruppo.
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky, che nella tabella non sono stati nemmeno annotati; inoltre, non tiene in considerazione i permessi legati al gruppo.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

FILE SORGENTI

'`lib/sys/stat.h`' [u0.13]

'`lib/sys/stat/chmod.c`' [i161.13.1]

'`lib/sys/stat/fchmod.c`' [i161.13.2]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs/path_chmod.c`' [i160.4.33]

VEDERE ANCHE

chmod(1) [u0.5], *chown(2)* [u0.5], *open(2)* [u0.28], *stat(2)* [u0.36].

os16: chown(2)

NOME

'**`chown`**', '**`fchown`**' - modifica della proprietà dei file

SINTASSI

```
#include <unistd.h>
int chown  (const char *path, uid_t uid, gid_t gid);
int fchown (int fdn, uid_t uid, gid_t gid);
```

DESCRIZIONE

Le funzioni ***chown()*** e ***fchown()***, modificano la proprietà di un file, fornendo il numero UID e il numero GID. Il file viene indicato, rispettivamente, attraverso il percorso scritto in una stringa, oppure come numero di descrittore già aperto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`EPERM`	Permessi insufficienti per eseguire l'operazione.
`ENOTDIR`	Uno dei componenti del percorso non è una directory.
`ENOENT`	Uno dei componenti del percorso non esiste.
`EBADF`	Il descrittore del file non è valido.

DIFETTI

Le funzioni consentono di attribuire il numero del gruppo, ma os16 non valuta i permessi di accesso ai file, relativi ai gruppi.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/chown.c`' [i161.17.4]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs/path_chown.c`' [i160.4.34]

'`kernel/fs/fd_chown.c`' [i160.4.2]

VEDERE ANCHE

chmod(2) [u0.4].

os16: clock(2)

NOME

'**`clock`**' - tempo della CPU espresso in unità '**`clock_t`**'

SINTASSI

```
#include <time.h>
clock_t clock (void);
```

DESCRIZIONE

La funzione ***clock()*** restituisce il tempo di utilizzo della CPU, espresso in unità '**`clock_t`**', corrispondenti a secondi diviso il valore della macro-variabile ***CLOCKS_PER_SEC***. Per os16, come dichiarato nel file '`time.h`', il valore di ***CLOCKS_PER_SEC*** è 18, essendo la frequenza di CPU poco più di 18 Hz.

VALORE RESTITUITO

La funzione restituisce il tempo di CPU, espresso in multipli di 1/18 di secondo.

FILE SORGENTI

'`lib/time.h`' [u0.16]

'`lib/time/clock.c`' [i161.16.2]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/k_libc/k_clock.c`' [i160.6.1]

VEDERE ANCHE

time(2) [u0.39].

os16: close(2)

NOME

'**close**' - chiusura di un descrittore di file

SINTASSI

```
#include <unistd.h>
int close (int fdn);
```

DESCRIZIONE

Le funzioni ***close()*** chiude un descrittore di file.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	Il descrittore del file non è valido.

FILE SORGENTI

'lib/unistd.h' [u0.17]
'lib/unistd/close.c' [i161.17.5]
'lib/sys/os16/sys.s' [i161.12.15]
'kernel/proc/_isr.s' [i160.9.1]
'kernel/proc/sysroutine.c' [i160.9.30]
'kernel/fs/fd_close.c' [i160.4.3]

VEDERE ANCHE

fcntl(2) [u0.13], *open(2)* [u0.28], *fclose(3)* [u0.27].

os16: dup(2)

NOME

'**dup**', '**dup2**' - duplicazione di descrittori di file

SINTASSI

```
#include <unistd.h>
int dup  (int fdn_old);
int dup2 (int fdn_old, int fdn_new);
```

DESCRIZIONE

Le funzioni ***dup()*** e ***dup2()*** servono a duplicare un descrittore di file. La funzione ***dup()*** duplica il descrittore ***fdn_old***, utilizzando il numero di descrittore libero più basso che sia disponibile; ***dup2()***, invece, richiede che il nuovo numero di descrittore sia specificato, attraverso il parametro ***fdn_new***. Tuttavia, se il numero di descrittore ***fdn_new*** risulta utilizzato, questo viene chiuso prima di diventare la copia di ***fdn_old***.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	Uno dei descrittori specificati non è valido.
EMFILE	Troppi file aperti per il processo.

FILE SORGENTI

'lib/unistd.h' [u0.17]
'lib/unistd/dup.c' [i161.17.6]
'lib/unistd/dup2.c' [i161.17.7]
'lib/sys/os16/sys.s' [i161.12.15]
'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]
'kernel/fs/fd_dup.c' [i160.4.4]
'kernel/fs/fd_dup2.c' [i160.4.5]

VEDERE ANCHE

close(2) [u0.7], *fcntl(2)* [u0.13], *open(2)* [u0.28].

os16: dup2(2)

Vedere *dup(2)* [u0.8].

os16: execve(2)

NOME

'**execve**' - esecuzione di un file

SINTASSI

```
#include <unistd.h>
int execve (const char *path, char *const argv[],
            char *const envp[]);
```

DESCRIZIONE

La funzione ***execve()*** è quella che avvia effettivamente un programma, mentre le altre funzioni '**exec...()**' offrono semplicemente un'interfaccia differente per l'avvio, ma poi si avvalgono di ***execve()*** per svolgere effettivamente quanto loro richiesto.

La funzione ***execve()*** avvia il file il cui percorso è specificato come stringa, nel primo argomento.

Il secondo argomento deve essere un array di stringhe, dove la prima deve rappresentare il nome del programma avviato e le successive sono gli argomenti da passare al programma. L'ultimo elemento di tale array deve essere un puntatore nullo, per poter essere riconosciuto.

Il terzo argomento deve essere un array di stringhe, rappresentanti l'ambiente da passare al nuovo processo. Per ambiente si intende l'insieme delle variabili di ambiente, pertanto queste stringhe devono avere la forma '***nome=valore***', per essere riconoscibili. Anche in questo caso, per poter individuare l'ultimo elemento dell'array, questo deve essere un puntatore nullo.

VALORE RESTITUITO

Se ***execve()*** riesce nel suo compito, non può restituire alcunché, dato che in quel momento, il processo chiamante viene rimpiazzato da quello del file che viene eseguito. Pertanto, se viene restituito qualcosa, può trattarsi solo di un valore che rappresenta un errore, ovvero -1, aggiornando anche la variabile ***errno*** di conseguenza.

ERRORI

Valore di ***errno***	Significato
E2BIG	Ci sono troppi argomenti.
ENOMEM	Memoria insufficiente.
ENOENT	Il file richiesto non esiste.
EACCES	Il file non può essere avviato per la mancanza dei permessi di acceso necessari.
ENOEXEC	Il file non può essere un file eseguibile, perché non ne ha le caratteristiche.
EIO	Errore di input-output.

DIFETTI

os16 non prevede l'interpretazione di script, perché non esiste alcun programma in grado di farlo. Anche la shell di os16 si limita a eseguire i comandi inseriti, ma non può interpretare un file.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/execle.c`’ [i161.17.10]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/proc/proc_sys_exec.c`’ [i160.9.21]

VEDERE ANCHE

fork(2) [u0.14], *exec(3)* [u0.20], *getopt(3)* [u0.52], *environ(7)* [u0.1].

os16: fchmod(2)

Vedere *chmod(2)* [u0.4].

os16: fchown(2)

Vedere *chown(2)* [u0.5].

os16: fcntl(2)

NOME

‘**`fcntl`**’ - configurazione e intervento sui descrittori di file

SINTASSI

```
#include <fcntl.h>
int fcntl (int fdn, int cmd, ...);
```

DESCRIZIONE

La funzione ***fcntl()*** esegue un’operazione, definita dal parametro ***cmd***, sul descrittore richiesto come primo parametro (***fdn***). A seconda del tipo di operazione richiesta, potrebbero essere necessari degli argomenti ulteriori, i quali però non possono essere formalizzati in modo esatto nel prototipo della funzione. Il valore del secondo parametro che rappresenta l’operazione richiesta, va fornito in forma di costante simbolica, come descritto nell’elenco seguente.

Sintassi	Descrizione
`fcntl (`***fdn***`, F_DUPFD, (int)` ***fdn_min***`)`	Richiede la duplicazione del descrittore di file ***fdn***, in modo tale che la copia abbia il numero di descrittore minore possibile, ma maggiore o uguale a quello indicato come argomento ***fdn_min***.
`fcntl (`***fdn***`, F_GETFD)` `fcntl (`***fdn***`, F_SETFD, (int)` ***fd_flags***`)`	Rispettivamente, legge o imposta, gli indicatori del descrittore di file ***fdn*** (eventualmente noti come *file descriptor flags* o solo *fd flags*). Per il momento, è possibile impostare un solo indicatore, ‘**`FD_CLOEXEC`**’, pertanto, al posto di ***fd_flags*** si può mettere solo la costante ‘**`FD_CLOEXEC`**’.
`fcntl (`***fdn***`, F_GETFL)` `fcntl (`***fdn***`, F_SETFL, (int)` ***fl_flags***`)`	Rispettivamente, legge o imposta, gli indicatori dello stato del file, relativi al descrittore ***fdn*** (eventualmente noti come *file flaga* o solo *fl flags*). Per impostare questi indicatori, vanno combinate delle costanti simboliche: ‘**`O_RDONLY`**’, ‘**`O_WRONLY`**’, ‘**`O_RDWR`**’, ‘**`O_CREAT`**’, ‘**`O_EXCL`**’, ‘**`O_NOCTTY`**’, ‘**`O_TRUNC`**’.

VALORE RESTITUITO

Il significato del valore restituito dalla funzione dipende dal tipo di operazione richiesta, come sintetizzato dalla tabella successiva.

Operazione richiesta	Significato del valore restituito
`F_DUPFD`	Si ottiene il numero del descrittore prodotto dalla copia, oppure `-1` in caso di errore.
`F_GETFD`	Si ottiene il valore degli indicatori del descrittore (*fd flags*), oppure `-1` in caso di errore.
`F_GETFL`	Si ottiene il valore degli indicatori del file (*fl flags*), oppure `-1` in caso di errore.

`F_GETOWN`
`F_SETOWN`
`F_GETLK`
`F_SETLK`
`F_SETLKW`
	Si ottiene `-1`, in quanto si tratta di operazioni non realizzate in questa versione della funzione, per os16.

altri tipi di operazione	Si ottiene 0 in caso di successo, oppure `-1` in caso di errore.

ERRORI

Valore di ***errno***	Significato
`E_NOT_IMPLEMENTED`	È stato richiesto un tipo di operazione che non è disponibile nel caso particolare di os16.
`EINVAL`	È stato richiesto un tipo di operazione non valido.

FILE SORGENTI

‘`lib/fcntl.h`’ [u0.4]

‘`lib/fcntl/fcntl.c`’ [i161.4.2]

VEDERE ANCHE

dup(2) [u0.8], *dup2(2)* [u0.8], *open(2)* [u0.28].

os16: fork(2)

NOME

‘**`fork`**’ - sdoppiamento di un processo, ovvero creazione di un processo figlio

SINTASSI

```
#include <unistd.h>
pid_t fork (void);
```

DESCRIZIONE

La funzione ***fork()*** crea una copia del processo in corso, la quale copia diventa un processo figlio del primo. Il processo figlio eredita una copia dei descrittori di file aperti e di conseguenza dei flussi di file e directory.

Il processo genitore riceve dalla funzione il valore del numero PID del processo figlio avviato; il processo figlio si mette a funzionare dal punto in cui si trova la funzione ***fork()***, restituendo però un valore nullo: in questo modo tale processo figlio può riconoscersi come tale.

VALORE RESTITUITO

La funzione restituisce al processo genitore il numero PID del processo figlio; al processo figlio restituisce zero. In caso di problemi, invece, il valore restituito è `-1` e la variabile ***errno*** risulta aggiornata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente per avviare un altro processo.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/fork.c`’ [i161.17.17]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/proc/proc_sys_fork.c’ [i160.9.23]

VEDERE ANCHE

execve(2) [u0.10], *wait(2)* [u0.43], *exec(3)* [u0.20].

os16: fstat(2)

Vedere *stat(2)* [u0.36].

os16: getcwd(2)

NOME

‘**getcwd**’ - determinazione della directory corrente

SINTASSI

```
#include <unistd.h>
char *getcwd (char *buffer, size_t size);
```

DESCRIZIONE

La funzione ***getcwd()*** modifica il contenuto dell’area di memoria a cui punta ***buffer***, copiandovi al suo interno la stringa che rappresenta il percorso della directory corrente. La scrittura all’interno di ***buffer*** può prolungarsi al massimo per ***size*** byte, incluso il codice nullo di terminazione delle stringhe.

VALORE RESTITUITO

La funzione restituisce il puntatore alla stringa che rappresenta il percorso della directory corrente, il quale deve coincidere con ***buffer***. In caso di errore, invece, la funzione restituisce il puntatore nullo ‘**NULL**’.

ERRORI

Valore di ***errno***	Significato
EINVAL	Il puntatore ***buffer*** non è valido.
E_LIMIT	Il percorso della directory corrente è troppo lungo, rispetto ai limiti realizzativi di os16.

DIFETTI

La funzione ***getcwd()*** di os16 deve comunicare con il kernel per ottenere l’informazione che le serve, perché la «u-area» (*User area*) è trattenuta all’interno del kernel stesso.

FILE SORGENTI

‘lib/unistd.h’ [u0.17]

‘lib/unistd/getcwd.c’ [i161.17.18]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

VEDERE ANCHE

chdir(2) [u0.3].

os16: geteuid(2)

Vedere *getuid(2)* [u0.18].

os16: getuid(2)

NOME

‘**getuid**’, ‘**geteuid**’ - determinazione dell’identità reale ed efficace

SINTASSI

```
#include <unistd.h>
uid_t getuid  (void);
uid_t geteuid (void);
```


DESCRIZIONE

La funzione ***getuid()*** restituisce il numero corrispondente all’identità reale del processo; la funzione ***geteuid()*** restituisce il numero dell’identità efficace del processo.

VALORE RESTITUITO

Il numero UID, reale o efficace del processo chiamante. Non sono previsti casi di errore.

DIFETTI

Le funzioni ***getuid()*** e ***geteuid()*** di os16 devono comunicare con il kernel per ottenere l’informazione che a loro serve, perché la «u-area» (*User area*) è trattenuta all’interno del kernel stesso.

FILE SORGENTI

‘lib/unistd.h’ [u0.17]

‘lib/unistd/getuid.c’ [i161.17.24]

‘lib/unistd/geteuid.c’ [i161.17.19]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

VEDERE ANCHE

setuid(2) [u0.33].

os16: getpgrp(2)

Vedere *getpid(2)* [u0.20].

os16: getpid(2)

NOME

‘**getpid**’, ‘**getppid**’, ‘**getpgrp**’ - determinazione del numero del processo o del gruppo di processi

SINTASSI

```
#include <unistd.h>
pid_t getpid  (void);
pid_t getppid (void);
pid_t getpgrp (void);
```

DESCRIZIONE

La funzione ***getpid()*** restituisce il numero del processo chiamante; la funzione ***getppid()*** restituisce il numero del processo genitore rispetto a quello chiamante; la funzione ***getpgrp()*** restituisce il numero attribuito al gruppo di processi a cui appartiene quello chiamante.

VALORE RESTITUITO

Il numero di processo o di gruppo di processi, relativo al contesto della funzione. Non sono previsti casi di errore.

DIFETTI

Le funzioni ***getpid()***, ***getppid()*** e ***getpgrp()*** di os16 devono comunicare con il kernel per ottenere l’informazione che a loro serve, perché la «u-area» (*User area*) è trattenuta all’interno del kernel stesso.

FILE SORGENTI

‘lib/unistd.h’ [u0.17]

‘lib/unistd/getpid.c’ [i161.17.22]

‘lib/unistd/getppid.c’ [i161.17.23]

‘lib/unistd/getpgrp.c’ [i161.17.21]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

VEDERE ANCHE

getuid(2) [u0.18] *fork(2)* [u0.14], *execve(2)* [u0.10].

os16: getppid(2)

Vedere *getpid(2)* [u0.20].

os16: kill(2)

NOME

'**kill**' - invio di un segnale a un processo

SINTASSI

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig)
```

DESCRIZIONE

La funzione ***kill()*** invia il segnale ***sig*** al processo numero ***pid***, oppure a un gruppo di processi. Questa realizzazione particolare di os16 comporta come segue:

- se il valore ***pid*** è maggiore di zero, il segnale viene inviato al processo con il numero ***pid***, ammesso di averne il permesso;
- se il valore ***pid*** è pari a zero, il segnale viene inviato a tutti i processi appartenenti allo stesso utente (quelli che hanno la stessa identità efficace, ovvero il valore ***euid***), ma se il processo che chiama la funzione lavora con un valore di ***euid*** pari a zero, il segnale viene inviato a tutti i processi, a partire dal numero due (si salta '**init**');
- valori negativi di ***pid*** non vengono presi in considerazione.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EPERM	Il processo non ha i permessi per inviare il segnale alla destinazione richiesta.
ESRCH	La ricerca del processo ***pid*** è fallita. Nel caso di os16, si ottiene questo errore anche per valori negativi di ***pid***.

FILE SORGENTI

'lib/sys/types.h' [u0.14]
'lib/signal.h' [u0.8]
'lib/signal/kill.c' [i161.8.1]
'lib/sys/os16/sys.s' [i161.12.15]
'kernel/proc/_isr.s' [i160.9.1]
'kernel/proc/sysroutine.c' [i160.9.30]
'kernel/proc/proc_sys_kill.c' [i160.9.24]

VEDERE ANCHE

signal(2) [u0.34].

os16: link(2)

NOME

'**link**' - creazione di un collegamento fisico tra un file esistente e un altro nome

SINTASSI

```
#include <unistd.h>
int link (const char *path_old, const char *path_new);
```

DESCRIZIONE

La funzione ***link()*** produce un nuovo collegamento a un file già esistente. Va fornito il percorso del file già esistente, ***path_old*** e quello del file da creare, in qualità di collegamento, ***path_new***. L'operazione può avvenire soltanto se i due percorsi si trovano sulla stessa unità di memorizzazione e se ci sono i permessi di scrittura necessari nella directory di destinazione. Dopo l'operazione di collegamento, fatta in questo modo, non è possibile distinguere quale sia il file originale e quale sia invece il nome aggiunto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione non consentita.
EEXIST	Il nome da creare esiste già.
EACCES	Accesso non consentito.
ENOENT	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
EROFS	Il file system consente soltanto un accesso in lettura.
ENOTDIR	Uno dei due percorsi non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.

FILE SORGENTI

'lib/unistd.h' [u0.17]
'lib/unistd/link.c' [i161.17.26]
'lib/sys/os16/sys.s' [i161.12.15]
'kernel/proc/_isr.s' [i160.9.1]
'kernel/proc/sysroutine.c' [i160.9.30]
'kernel/fs/path_link.c' [i160.4.40]

VEDERE ANCHE

ln(1) [u0.11] *open(2)* [u0.28], *stat(2)* [u0.36], *unlink(2)* [u0.42].

os16: lseek(2)

NOME

'**lseek**' - riposizionamento dell'indice di accesso a un descrittore di file

SINTASSI

```
#include <unistd.h>
off_t lseek (int fdn, off_t offset, int whence);
```

DESCRIZIONE

La funzione ***lseek()*** consente di riposizionare l'indice di accesso interno al descrittore di file ***fdn***. Per fare questo occorre prima determinare un punto di riferimento, rappresentato dal parametro ***whence***, dove va usata una macro-variabile definita nel file 'unistd.h'. Può trattarsi dei casi seguenti.

Valore di ***whence***	Significato
SEEK_SET	Lo scostamento si riferisce all'inizio del file.
SEEK_CUR	Lo scostamento si riferisce alla posizione che ha già l'indice interno al file.
SEEK_END	Lo scostamento si riferisce alla fine del file.

Lo scostamento indicato dal parametro ***offset*** si applica a partire dalla posizione a cui si riferisce ***whence***, pertanto può avere segno positivo o negativo, ma in ogni caso non è possibile collocare l'indice prima dell'inizio del file.

VALORE RESTITUITO

Se l'operazione avviene con successo, la funzione restituisce il valore dell'indice riposizionato, preso come scostamento a partire dall'inizio del file. In caso di errore, restituisce invece il valore -1, aggiornando di conseguenza anche la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il valore di ***whence*** non è contemplato, oppure la combinazione tra ***whence*** e ***offset*** non è valida.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/lseek.c`’ [i161.17.27]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/fd_lseek.c`’ [i160.4.7]

VEDERE ANCHE

dup(2) [u0.8] *fork(2)* [u0.14], *open(2)* [u0.28], *fseek(3)* [u0.43].

os16: mkdir(2)

NOME

‘**`mkdir`**’ - creazione di una directory

SINTASSI

```
#include <sys/stat.h>
int mkdir (const char *path, mode_t mode);
```

DESCRIZIONE

La funzione ***mkdir()*** crea una directory, indicata attraverso un percorso, nel parametro ***path***, specificando la modalità dei permessi, con il parametro ***mode***.

Tuttavia, il valore del parametro ***mode*** non viene preso in considerazione integralmente: di questo si considerano solo gli ultimi nove bit, ovvero quelli dei permessi di utenti, gruppi e altri utenti; inoltre, vengono tolti i bit presenti nella maschera dei permessi associata al processo (si veda anche *umask(2)* [u0.40]).

La directory che viene creata in questo modo, appartiene all’identità efficace del processo, ovvero all’utente per conto del quale questo sta funzionando.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso della directory da creare, non è una directory.
`ENOENT`	Una porzione del percorso della directory da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]

‘`lib/sys/stat/mkdir.c`’ [i161.13.4]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/path_mkdir.c`’ [i160.4.41]

VEDERE ANCHE

mkdir(1) [u0.15], *chmod(2)* [u0.4], *chown(2)* [u0.5], *mknod(2)* [u0.26], *mount(2)* [u0.27], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42].

os16: mknod(2)

NOME

‘**`mknod`**’ - creazione di un file vuoto di qualunque tipo

SINTASSI

```
#include <sys/stat.h>
int mknod (const char *path, mode_t mode, dev_t device);
```

DESCRIZIONE

La funzione ***mknod()*** crea un file vuoto, di qualunque tipo. Potenzialmente può creare anche una directory, ma priva di qualunque voce, rendendola così non adeguata al suo scopo (una directory richiede almeno le voci ‘`.`’ e ‘`..`’, per potersi considerare tale).

Il parametro ***path*** specifica il percorso del file da creare; il parametro ***mode*** serve a indicare il tipo di file da creare, oltre ai permessi comuni.

Il parametro ***device***, con il quale va indicato il numero di un dispositivo (completo di numero primario e secondario), viene preso in considerazione soltanto se nel parametro ***mode*** si richiede la creazione di un file di dispositivo a caratteri o a blocchi.

Il valore del parametro ***mode*** va costruito combinando assieme delle macro-variabili definite nel file ‘`sys/stat.h`’, come descritto nella pagina di manuale *stat(2)* [u0.36], tenendo conto che os16 non può gestire file FIFO, collegamenti simbolici e socket di dominio Unix.

Il valore del parametro ***mode***, per la porzione che riguarda i permessi di accesso al file, viene comunque filtrato con la maschera dei permessi (*umask(2)* [u0.36]).

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso del file da creare, non è una directory.
`ENOENT`	Una porzione del percorso del file da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]

‘`lib/sys/stat/mknod.c`’ [i161.13.5]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/path_mknod.c`’ [i160.4.42]

VEDERE ANCHE

mkdir(2) [u0.25], *chmod(2)* [u0.4], *chown(2)* [u0.5], *fcntl(2)* [u0.13], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42].

os16: mount(2)

NOME

‘**`mount`**’, ‘**`umount`**’ - innesto e distacco di unità di memorizzazione

SINTASSI

```
#include <sys/os16.h>
int mount  (const char *path_dev, const char *path_mnt,
            int options);
int umount (const char *path_mnt);
```

DESCRIZIONE

La funzione ***mount()*** permette l'innesto di un'unità di memorizzazione individuata attraverso il percorso del file di dispositivo nel parametro ***path_dev***, nella directory corrispondente al percorso ***path_mnt***, con le opzioni indicate numericamente nell'ultimo argomento ***options***. Le opzioni di innesto, rappresentate attraverso delle macro-variabili, sono solo due:

Opzione	Descrizione
`MOUNT_DEFAULT`	Innesto normale, in lettura e scrittura.
`MOUNT_RO`	Innesto in sola lettura.

La funzione ***umount()*** consente di staccare un innesto fatto precedentemente, specificando il percorso della directory in cui questo è avvenuto.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: va verificato il contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Problema di accesso dovuto alla mancanza dei permessi necessari.
`ENOTDIR`	Ciò che dovrebbe essere una directory, non lo è.
`EBUSY`	La directory innesta già un file system e non può innestarne un altro.
`ENOENT`	La directory non esiste.
`E_NOT_MOUNTED`	La directory non innesta un file system da staccare.
`EUNKNOWN`	Si è verificato un problema non previsto e sconosciuto.

FILE SORGENTI

'`lib/sys/os16.h`' [u0.12]

'`lib/sys/os16/mount.c`' [i161.12.12]

'`lib/sys/os16/umount.c`' [i161.12.16]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs/path_mount.c`' [i160.4.43]

'`kernel/fs/path_umount.c`' [i160.4.45]

VEDERE ANCHE

mount(8) [u0.4], *umount(8)* [u0.4].

os16: open(2)

«

NOME

'**open**' - apertura di un file puro e semplice oppure di un file di dispositivo

SINTASSI

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open (const char *path, int oflags);
int open (const char *path, int oflags, mode_t mode);
```


DESCRIZIONE

La funzione ***open()*** apre un file, indicato attraverso il percorso ***path***, in base alle opzioni rappresentate dagli indicatori ***oflags***. A seconda del tipo di indicatori specificati, potrebbe essere richiesto il parametro ***mode***.

Quando la funzione porta a termine correttamente il proprio compito, restituisce il numero del descrittore del file associato, il quale è sempre quello di valore più basso disponibile per il processo elaborativo in corso.

Il descrittore di file ottenuto inizialmente con la funzione ***open()***, è legato al processo elaborativo in corso; tuttavia, se successivamente il processo si sdoppia attraverso la funzione ***fork()***, tale descrittore, se ancora aperto, viene duplicato nella nuova copia del processo. Inoltre, se per il descrittore aperto non viene impostato l'indicatore '**`FD_CLOEXEC`**' (con l'ausilio della funzione ***fcntl()***), se il processo viene rimpiazzato con la funzione ***execve()***, il descrittore aperto viene ereditato dal nuovo processo.

Il parametro ***oflags*** richiede necessariamente la specificazione della modalità di accesso, attraverso la combinazione appropriata dei valori: '**`O_RDONLY`**', '**`O_WRONLY`**', '**`O_RDWR`**'. Inoltre, si possono combinare altri indicatori: '**`O_CREAT`**', '**`O_TRUNC`**', '**`O_APPEND`**'.

Opzione	Descrizione
`O_RDONLY`	Richiede un accesso in lettura.
`O_WRONLY`	Richiede un accesso in scrittura.
`O_RDWR` `O_RDONLY|O_WRONLY`	Richiede un accesso in lettura e scrittura (la combinazione di '**`R_RDONLY`**' e di '**`O_WRONLY`**' è equivalente all'uso di '**`O_RDWR`**').
`O_CREAT`	Richiede di creare contestualmente il file, ma in tal caso va usato anche il parametro ***mode***.
`O_TRUNC`	Se file da aprire esiste già, richiede che questo sia ridotto preventivamente a un file vuoto.
`O_APPEND`	Fa in modo che le operazioni di scrittura avvengano sempre partendo dalla fine del file.

Quando si utilizza l'opzione '**`O_CREAT`**', è necessario stabilire la modalità dei permessi, cosa che va fatta preferibilmente attraverso la combinazione di costanti simboliche appropriate, come elencato nella tabella successiva. Tale combinazione va fatta con l'uso dell'operatore OR binario; per esempio: '**`S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH`**'. Va osservato che os16 non gestisce i gruppi di utenti, pertanto, la definizione dei permessi relativi agli utenti appartenenti al gruppo proprietario di un file, non ha poi effetti pratici nel controllo degli accessi per tale tipo di contesto.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	00700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	00400_8	Lettura per l'utente proprietario.
`S_IWUSR`	00200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	00100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXG`	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	00040_8	Lettura per il gruppo.
`S_IWGRP`	00020_8	Scrittura per il gruppo.
`S_IXGRP`	00010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

VALORE RESTITUITO

La funzione restituisce il numero del descrittore del file aperto, se l'operazione ha avuto successo, altrimenti dà semplicemente `-1`, impostando di conseguenza il valore della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/sys/stat.h`' [u0.13]

'`lib/fcntl.h`' [u0.4]

'`lib/fcntl/open.c`' [i161.4.3]

VEDERE ANCHE

chmod(2) [u0.4], *chown(2)* [u0.5], *close(2)* [u0.7], *dup(2)* [u0.8], *fcntl(2)* [u0.13], *link(2)* [u0.24], *mknod(2)* [u0.26], *mount(2)* [u0.27], *read(2)* [u0.29], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42], *write(2)* [u0.44], *fopen(3)* [u0.35].

os16: read(2)

«

NOME

'**`read`**' - lettura di descrittore di file

SINTASSI

```
#include <unistd.h>
ssize_t read (int fdn, void *buffer, size_t count);
```

DESCRIZIONE

La funzione ***read()*** cerca di leggere il file rappresentato dal descrittore ***fdn***, partendo dalla posizione in cui si trova l'indice interno di accesso, per un massimo di ***count*** byte, collocando i dati letti in memoria a partire dal puntatore ***buffer***. L'indice interno al file viene fatto avanzare della quantità di byte letti effettivamente, se invece si incontra la fine del file, viene aggiornato l'indicatore interno per segnalare tale fatto.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti effettivamente, oppure zero se è stata raggiunta la fine del file e non si può proseguire oltre. Va osservato che la lettura effettiva di una quantità inferiore di byte rispetto a quanto richiesto non costituisce un errore: in quel caso i byte mancanti vanno richiesti con successive operazioni di lettura. In caso di errore, la funzione restituisce il valore `-1`, aggiornando contestualmente la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in lettura.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os16.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/read.c`' [i161.17.28]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/fs/fd_read.c`' [i160.4.9]

VEDERE ANCHE

close(2) [u0.7] *open(2)* [u0.28], *write(2)* [u0.44].

os16: rmdir(2)

«

NOME

'**`rmdir`**' - eliminazione di una directory vuota

SINTASSI

```
#include <unistd.h>
int rmdir (const char *path);
```

DESCRIZIONE

La funzione ***rmdir()*** cancella la directory indicata come percorso, nella stringa ***path***, purché sia vuota.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
`-1`	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso ***path*** non è valido o è semplicemente un puntatore nullo.
`ENOTDIR`	Il nome indicato o le posizioni intermedie del percorso si riferiscono a qualcosa che non è una directory.
`ENOTEMPTY`	La directory che si vorrebbe cancellare non è vuota.
`EROFS`	La directory si trova in un'unità innestata in sola lettura.
`EPERM`	Mancano i permessi necessari per eseguire l'operazione.
`EUNKNOWN`	Si è verificato un errore imprevisto e sconosciuto.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/rmdir.c`' [i161.17.29]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/fs/path_unlink.c`' [i160.4.46]

VEDERE ANCHE

mkdir(2) [u0.25], *unlink(2)* [u0.42].

os16: seteuid(2)

«

Vedere *setuid(2)* [u0.33].

os16: setpgrp(2)

NOME

'**setpgrp**' - impostazione del gruppo a cui appartiene il processo

SINTASSI

```
#include <unistd.h>
int setpgrp (void);
```

DESCRIZIONE

La funzione ***setpgrp()*** fa sì che il processo in corso costituisca un proprio gruppo autonomo, corrispondente al proprio numero PID. In altri termini, la funzione serve per iniziare un nuovo gruppo di processi, a cui i processi figli creati successivamente vengano associati in modo predefinito.

VALORE RESTITUITO

La funzione termina sempre con successo e restituisce sempre zero.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/setpgrp.c`' [i161.17.31]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

VEDERE ANCHE

getpgrp(2) [u0.32], *getuid(2)* [u0.33].

os16: setuid(2)

NOME

'**setuid**', '**seteuid**' - impostazione dell'identità dell'utente

SINTASSI

```
#include <unistd.h>
int setuid  (uid_t uid);
int seteuid (uid_t uid);
```

DESCRIZIONE

A ogni processo viene attribuita l'identità di un utente, rappresentata da un numero, noto come UID, ovvero *user identity*. Tuttavia si distinguono tre tipi di numeri UID: l'identità reale, l'identità efficace e un'identità salvata in precedenza. L'identità efficace (EUID) è quella con cui opera sostanzialmente il processo; l'identità salvata è quella che ha avuto il processo in un altro momento in qualità di identità efficace e che per qualche motivo non ha più.

La funzione ***setuid()*** riceve come argomento un numero UID e si comporta diversamente a seconda della personalità del processo, come descritto nell'elenco successivo:

- se l'identità efficace del processo corrisponde a zero, trattandosi di un caso particolarmente privilegiato, tutte le identità del processo (reale, efficace e salvata) vengono inizializzate con il valore fornito alla funzione ***setuid()***;
- se l'identità efficace del processo corrisponde a quella fornita come argomento a ***setuid()***, nulla cambia nella gestione delle identità del processo;
- se l'identità reale o quella salvata in precedenza corrispondono a quella fornita come argomento di ***setuid()***, viene aggiornato il valore dell'identità efficace, senza cambiare le altre;
- in tutti gli altri casi, l'operazione non è consentita e si ottiene un errore.

La funzione ***seteuid()*** riceve come argomento un numero UID e imposta con tale valore l'identità efficace del processo, purché si verifichi almeno una delle condizioni seguenti:

- l'identità efficace del processo è zero;
- l'identità reale o quella salvata del processo corrisponde all'identità efficace che si vuole impostare;
- l'identità efficace del processo corrisponde già all'identità efficace che si vuole impostare.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Non si dispone dei permessi necessari a eseguire il cambiamento di identità.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/setuid.c`' [i161.17.32]

'`lib/unistd/seteuid.c`' [i161.17.30]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

VEDERE ANCHE

getuid(2) [u0.33].

os16: signal(2)

NOME

'**signal**' - abilitazione e disabilitazione dei segnali

SINTASSI

```
#include <signal.h>
sighandler_t signal (int sig, sighandler_t handler);
```

DESCRIZIONE

La funzione ***signal()*** di os16 consente soltanto di abilitare o disabilitare un segnale, il quale, se abilitato, può essere gestito solo in modo predefinito dal sistema. Pertanto, il valore che può assumere ***handler*** sono solo: '**SIG_DFL**' (gestione predefinita) e '**SIG_IGN**' (ignora il segnale). Il primo parametro, ***sig***, rappresenta il segnale a cui applicare ***handler***.

Il tipo '**sighandler_t**' è definito nel file '`signal.h`', nel modo seguente:

```
typedef void (*sighandler_t) (int);
```

Rappresenta il puntatore a una funzione avente un solo parametro di tipo '**int**', la quale non restituisce alcunché.

VALORE RESTITUITO

La funzione restituisce il tipo di «gestione» impostata precedentemente per il segnale richiesto, ovvero '**SIG_ERR**' in caso di errore.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il numero ***sig*** o il valore di ***handler*** non sono validi.

NOTE

Lo scopo della funzione ***signal()*** dovrebbe essere quello di consentire l'associazione di un evento, manifestato da un segnale, all'esecuzione di un'altra funzione, avente una forma del tipo '***nome*** **(int)**'. os16 non consente tale gestione, pertanto lascia soltanto la possibilità di attribuire un comportamento predefinito al segnale scelto, oppure di disabilitarlo, ammesso che ciò sia consentito. Sotto questo aspetto, il fatto di dover gestire i valori '**SIG_ERR**', '**SIG_DFL**' e '**SIG_IGN**', come se fossero puntatori

a una funzione, diventa superfluo, ma rimane utile soltanto per mantenere un minimo di conformità con quello che è lo standard della funzione ***signal()***.

FILE SORGENTI

‘`lib/signal.h`’ [u0.8]

‘`lib/signal/signal.c`’ [i161.8.2]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/proc/proc_sys_signal.c`’ [i160.9.27]

VEDERE ANCHE

kill(2) [u0.22].

os16: sleep(2)

NOME

‘**`sleep`**’ - pausa volontaria del processo chiamante

SINTASSI

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds);
```

DESCRIZIONE

La funzione ***sleep()*** chiede di mettere a riposo il processo chiamante per la quantità di secondi indicata come argomento. Il processo può però essere risvegliato prima della conclusione di tale durata, ma in tal caso la funzione restituisce la quantità di secondi che non sono stati usati per il «riposo» del processo.

VALORE RESTITUITO

La funzione restituisce zero se la pausa richiesta è trascorsa completamente; altrimenti, restituisce quanti secondi mancano ancora per completare il tempo di riposo chiesto originariamente. Non si prevede il manifestarsi di errori.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/sleep.c`’ [i161.17.33]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

VEDERE ANCHE

signal(2) [u0.34].

os16: stat(2)

NOME

‘**`stat`**’, ‘**`fstat`**’ - interrogazione dello stato di un file

SINTASSI

```
#include <sys/stat.h>
int stat  (const char *path, struct stat *buffer);
int fstat (int fdn, struct stat *buffer);
```

DESCRIZIONE

Le funzioni ***stat()*** e ***fstat()*** interrogano il sistema su di un file, per ottenerne le caratteristiche in forma di variabile strutturata di tipo ‘**`struct stat`**’.

La funzione ***stat()*** individua il file attraverso una stringa contenente il suo percorso (***path***); la funzione ‘**`fstat`**’ si riferisce a un file aperto di cui si conosce il numero del descrittore (***fdn***). In entrambi i casi, la struttura che deve accogliere l’esito dell’interrogazione, viene indicata attraverso un puntatore, come ultimo argomento (***buffer***).

La struttura ‘**`struct stat`**’ è definita nel file ‘`sys/stat.h`’ nel modo seguente:


```
struct stat {
  dev_t     st_dev;     // Device containing the file.
  ino_t     st_ino;     // File serial number (inode
                        // number).
  mode_t    st_mode;    // File type and permissions.
  nlink_t   st_nlink;   // Links to the file.
  uid_t     st_uid;     // Owner user id.
  gid_t     st_gid;     // Owner group id.
  dev_t     st_rdev;    // Device number if it is a
                        // device file.
  off_t     st_size;    // File size.
  time_t    st_atime;   // Last access time.
  time_t    st_mtime;   // Last modification time.
  time_t    st_ctime;   // Last inode modification.
  blksize_t st_blksize; // Block size for I/O operations.
  blkcnt_t  st_blocks;  // File size / block size.
};
```

Va osservato che il file system Minix 1, usato da os16, riporta esclusivamente la data e l’ora di modifica, pertanto le altre due date previste sono sempre uguali a quella di modifica.

Il membro ***st_mode***, oltre alla modalità dei permessi che si cambiano con *chmod(2)* [u0.4], serve ad annotare altre informazioni. Nel file ‘`sys/stat.h`’ sono definite delle macroistruzioni, utili per individuare il tipo di file. Queste macroistruzioni si risolvono in un valore numerico diverso da zero, solo se la condizione che rappresentano è vera:

Macroistruzione	Significato
`S_ISBLK (`***m***`)`	È un file di dispositivo a blocchi?
`S_ISCHR (`***m***`)`	È un file di dispositivo a caratteri?
`S_ISFIFO (`***m***`)`	È un file FIFO?
`S_ISREG (`***m***`)`	È un file puro e semplice?
`S_ISDIR (`***m***`)`	È una directory?
`S_ISLNK (`***m***`)`	È un collegamento simbolico?
`S_ISSOCK (`***m***`)`	È un socket di dominio Unix?

Naturalmente, anche se nel file system possono esistere file di ogni tipo, poi os16 non è in grado di gestire i file FIFO, i collegamenti simbolici e nemmeno i socket di dominio Unix.

Nel file ‘`sys/stat.h`’ sono definite anche delle macro-variabili per individuare e facilitare la selezione dei bit che compongono le informazioni del membro ***st_mode***:

Modalità simbolica	Modalità numerica	Descrizione
`S_IFMT`	0170000_8	Maschera che raccoglie tutti i bit che individuano il tipo di file.
`S_IFBLK`	0060000_8	File di dispositivo a blocchi.
`S_IFCHR`	0020000_8	File di dispositivo a caratteri.
`S_IFIFO`	0010000_8	File FIFO.
`S_IFREG`	0100000_8	File puro e semplice.
`S_IFDIR`	0040000_8	Directory.
`S_IFLNK`	0120000_8	Collegamento simbolico.
`S_IFSOCK`	0140000_8	Socket di dominio Unix.

Modalità simbolica	Modalità numerica	Descrizione
`S_ISUID`	0004000_8	SUID.
`S_ISGID`	0002000_8	SGID.
`S_ISVTX`	0001000_8	Sticky.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	0000700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	0000400_8	Lettura per l'utente proprietario.
`S_IWUSR`	0000200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	0000100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXG`	0000070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	0000040_8	Lettura per il gruppo.
`S_IWGRP`	0000020_8	Scrittura per il gruppo.
`S_IXGRP`	0000010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	0000007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	0000004_8	Lettura per gli altri utenti.
`S_IWOTH`	0000002_8	Scrittura per gli altri utenti.
`S_IXOTH`	0000001_8	Esecuzione per gli altri utenti.

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky; inoltre, non tiene in considerazione i permessi legati al gruppo, perché non ne tiene traccia.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
`-1`	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Troppi file aperti nel sistema.
`ENOENT`	File non trovato.
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]
‘`lib/sys/stat/stat.c`’ [i161.13.6]
‘`lib/sys/stat/fstat.c`’ [i161.13.3]
‘`lib/sys/os16/sys.s`’ [i161.12.15]
‘`kernel/proc/_isr.s`’ [i160.9.1]
‘`kernel/proc/sysroutine.c`’ [i160.9.30]
‘`kernel/fs/path_stat.c`’ [i160.4.44]
‘`kernel/fs/fd_stat.c`’ [i160.4.11]

VEDERE ANCHE

chmod(2) [u0.4], *chown(2)* [u0.5].

os16: sys(2)

NOME

‘**`sys`**’ - chiamata di sistema

SINTASSI

```
#include <sys/os16.h>
void sys (int syscallnr, void *message, size_t size);
```


DESCRIZIONE

Attraverso la funzione ***sys()*** si effettuano tutte le chiamate di sistema, passando al kernel un messaggio, a cui punta ***message***, lungo ***size*** byte. A seconda dei casi, il messaggio può essere modificato dal kernel, come risposta alla chiamata.

Il messaggio è in pratica una variabile strutturata, la cui articolazione cambia a seconda del tipo di chiamata, pertanto si rende necessario specificarne ogni volta la dimensione.

Le strutture usate per comporre i messaggi hanno alcuni membri ricorrenti frequentemente:

Membro	Descrizione
`char path[PATH_MAX];`	Un percorso di file o directory.
`… ret;`	Serve a contenere il valore restituito dalla funzione che nel kernel compie effettivamente il lavoro. Il tipo del membro varia caso per caso.
`int errno;`	Serve a contenere il numero dell'errore prodotto dalla funzione che nel kernel compie effettivamente il lavoro.
`int errln;`	Serve a contenere il numero della riga di codice in cui si è prodotto un errore.
`char errfn[PATH_MAX];`	Serve a contenere il nome del file in cui si è prodotto un errore.

Le funzioni che si avvalgono di ***sys()***, prima della chiamata devono provvedere a compilare il messaggio con i dati necessari, dopo la chiamata devono acquisire i dati di ritorno, contenuti nel messaggio aggiornato dal kernel, e in particolare devono aggiornare le variabili ***errno***, ***errln*** e ***errfl***, utilizzando i membri con lo stesso nome presenti nel messaggio.

FILE SORGENTI

‘`lib/sys/os16.h`’ [u0.12]
‘`lib/sys/os16/sys.s`’ [i161.12.15]
‘`kernel/proc/_isr.s`’ [i160.9.1]
‘`kernel/proc/sysroutine.c`’ [i160.9.30]

os16: stime(2)

Vedere *time(2)* [u0.39].

os16: time(2)

NOME

‘**`time`**’, ‘**`stime`**’ - lettura o impostazione della data e dell'ora del sistema

SINTASSI

```
#include <time.h>
time_t time  (time_t *timer);
int    stime (time_t *timer);
```

DESCRIZIONE

La funzioni ***time()*** legge la data e l'ora attuale del sistema, espressa in secondi; se il puntatore ***timer*** è valido (non è ‘**`NULL`**’), il risultato dell'interrogazione viene salvato anche in ciò a cui questo punta.

La funzione ***stime()*** consente di modificare la data e l'ora attuale del sistema, fornendo il puntatore alla variabile contenente la quantità di secondi trascorsi a partire dall'ora zero del 1 gennaio 1970.

VALORE RESTITUITO

La funzione ***time()*** restituisce la data e l'ora attuale del sistema, espressa in secondi trascorsi a partire dall'ora zero del 1 gennaio 1970.

La funzione ***stime()*** restituisce zero in caso di successo e, teoricamente, `-1` in caso di errore, ma attualmente nessun tipo di errore è previsto.

DIFETTI

La funzione ***stime()*** dovrebbe essere riservata a un utente privilegiato, mentre attualmente qualunque utente può servirsene per cambiare la data di sistema.

FILE SORGENTI

‘`lib/time.h`’ [u0.16]

‘`lib/time/time.c`’ [i161.16.6]

‘`lib/time/stime.c`’ [i161.16.5]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/k_libc.h`’ [u0.6]

‘`kernel/k_libc/k_time.c`’ [i160.6.11]

‘`kernel/k_libc/k_stime.c`’ [i160.6.10]

VEDERE ANCHE

date(1) [u0.8], *ctime(3)* [u0.13].

os16: umask(2)

NOME

‘**`umask`**’ - maschera dei permessi

SINTASSI

```
#include <sys/stat.h>
mode_t umask (mode_t mask);
```

DESCRIZIONE

La funzione ***umask()*** modifica la maschera dei permessi associata al processo in corso. Nel contenuto del parametro ***mask*** vengono presi in considerazione soltanto i nove bit meno significativi, i quali rappresentano i premessi di accesso di utente proprietario, gruppo e altri utenti.

La maschera dei permessi viene usata dalle funzioni che creano dei file, per limitare i permessi in modo automatico: ciò che appare attivo nella maschera è quello che non viene consentito nella creazione del file.

VALORE RESTITUITO

La funzione restituisce il valore che aveva la maschera dei permessi prima della chiamata.

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]

‘`lib/sys/stat/umask.c`’ [i161.13.7]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

VEDERE ANCHE

mkdir(2) [u0.25], *chmod(2)* [u0.4], *open(2)* [u0.28], *stat(2)* [u0.36].

os16: umount(2)

Vedere *mount(2)* [u0.27].

os16: unlink(2)

NOME

‘**`unlink`**’ - cancellazione di un nome

SINTASSI

```
#include <unistd.h>
int unlink (const char *path);
```


DESCRIZIONE

La funzione ***unlink()*** cancella un nome da una directory, ma se si tratta dell’ultimo collegamento che ha quel file, allora libera anche l’inode corrispondente.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`ENOTEMPTY`	È stata tentata la cancellazione di una directory, ma questa non è vuota.
`ENOTDIR`	Una delle directory del percorso, non è una directory.
`ENOENT`	Il nome richiesto non esiste.
`EROFS`	Il file system è in sola lettura.
`EPERM`	Mancano i permessi necessari.
`EUNKNOWN`	Si è verificato un errore imprevisto e sconosciuto.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/unlink.c`’ [i161.17.35]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/path_unlink.c`’ [i160.4.46]

VEDERE ANCHE

rm(1) [u0.18], *link(2)* [u0.23], *rmdir(2)* [u0.30].

os16: wait(2)

NOME

‘**`wait`**’ - attesa della morte di un processo figlio

SINTASSI

```
#include <sys/wait.h>
pid_t wait (int *status);
```

DESCRIZIONE

La funzione ***wait()*** mette il processo in pausa, in attesa della morte di un processo figlio; quando ciò dovesse accadere, il valore di ****status*** verrebbe aggiornato con quanto restituito dal processo defunto e il processo sospeso riprenderebbe l’esecuzione.

VALORE RESTITUITO

La funzione restituisce il numero del processo defunto, oppure -1 se non ci sono processi figli.

ERRORI

Valore di ***errno***	Significato
`ECHILD`	Non ci sono processi figli da attendere.

FILE SORGENTI

‘`lib/sys/wait.h`’ [u0.15]

‘`lib/sys/wait/wait.c`’ [i161.15.1]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/proc/proc_sys_wait.c`’ [i160.9.28]

VEDERE ANCHE

_exit(2) [u0.2], *fork(2)* [u0.14], *kill(2)* [u0.22], *signal(2)* [u0.34].

os16: write(2)

NOME

'**write**' - scrittura di un descrittore di file

SINTASSI

```
#include <unistd.h>
ssize_t write (int fdn, const void *buffer, size_t count);
```

DESCRIZIONE

La funzione ***write()*** consente di scrivere fino a un massimo di ***count*** byte, tratti dall'area di memoria che inizia all'indirizzo ***buffer***, presso il file rappresentato dal descrittore ***fdn***. La scrittura avviene a partire dalla posizione in cui si trova l'indice interno.

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti effettivamente e in tal caso è possibile anche ottenere una quantità pari a zero. Se si verifica invece un errore, la funzione restituisce -1 e aggiorna la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	Il numero del descrittore di file non è valido.
EINVAL	Il file non è aperto in scrittura.
EISDIR	Il file è una directory.
E_FILE_TYPE_UNSUPPORTED	Il file è di tipo non gestibile con os16.
EIO	Errore di input-output.

FILE SORGENTI

'lib/unistd.h' [u0.17]

'lib/unistd/write.c' [i161.17.36]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/fs/fd_write.c' [i160.4.12]

VEDERE ANCHE

close(2) [u0.7], *lseek(2)* [u0.24], *open(2)* [u0.28], *read(2)* [u0.29], *fwrite(3)* [u0.48].

os16: z(2)

NOME

'**z**_...' - funzioni provvisorie

SINTASSI

```
#include <sys/os16.h>
void z_perror   (const char *string);
int  z_printf   (char *format, ...);
int  z_putchar  (int c);
int  z_puts     (char *string);
int  z_vprintf  (char *format, va_list arg);
```

DESCRIZIONE

Le funzioni del gruppo '**z**_...**()**' eseguono compiti equivalenti a quelli delle funzioni di libreria con lo stesso nome, ma prive del prefisso '**z**_'. Queste funzioni '**z**_...**()**' si avvalgono, per il loro lavoro, di chiamate di sistema particolari; la loro realizzazione si è resa necessaria durante lo sviluppo di os16, prima che potesse essere disponibile un sistema di gestione centralizzato dei dispositivi.

Queste funzioni non sono più utili, ma rimangono per documentare le fasi realizzative iniziali di os16 e, d'altro canto, possono servire se si rende necessario raggirare la gestione dei dispositivi per visualizzare dei messaggi sullo schermo.

FILE SORGENTI

'lib/sys/os16.h' [u0.12]

'lib/sys/os16/z_perror.c' [i161.12.17]

'lib/sys/os16/z_printf.c' [i161.12.18]

'lib/sys/os16/z_putchar.c' [i161.12.19]

'lib/sys/os16/z_puts.c' [i161.12.20]

'lib/sys/os16/z_vprintf.c' [i161.12.21]

VEDERE ANCHE

perror(3) [u0.77], *printf(3)* [u0.78], *putchar(3)* [u0.37], *puts(3)* [u0.38], *vprintf(3)* [u0.128], *vsprintf(3)* [u0.128].

os16: z_perror(2)

Vedere *z(2)* [u0.45].

os16: z_printf(2)

Vedere *z(2)* [u0.45].

os16: z_putchar(2)

Vedere *z(2)* [u0.45].

os16: z_puts(2)

Vedere *z(2)* [u0.45].

os16: z_vprintf(2)

Vedere *z(2)* [u0.45].

os16: access(3) 1434
os16: abort(3) 1435
os16: abs(3) 1435
os16: atexit(3) 1435
os16: atoi(3) 1436
os16: atol(3) 1437
os16: basename(3) 1437
os16: bp(3) 1437
os16: clearerr(3) 1437
os16: closedir(3) 1438
os16: creat(3) 1438
os16: cs(3) 1438
os16: ctime(3) 1439
os16: dirname(3) 1440
os16: div(3) 1440
os16: ds(3) 1441
os16: endpwent(3) 1441
os16: errno(3) 1441
os16: es(3) 1445
os16: exec(3) 1445
os16: execl(3) 1446
os16: execle(3) 1446
os16: execlp(3) 1446
os16: execv(3) 1446
os16: execvp(3) 1446
os16: exit(3) 1446
os16: fclose(3) 1446
os16: feof(3) 1447
os16: ferror(3) 1447
os16: fflush(3) 1448
os16: fgetc(3) 1448
os16: fgetpos(3) 1448
os16: fgets(3) 1449
os16: fileno(3) 1450
os16: fopen(3) 1451
os16: fprintf(3) 1452
os16: fputc(3) 1452
os16: fputs(3) 1452
os16: fread(3) 1453
os16: free(3) 1453
os16: freopen(3) 1453
os16: fscanf(3) 1453
os16: fseek(3) 1453
os16: fseeko(3) 1454
os16: fsetpos(3) 1454
os16: ftell(3) 1454
os16: ftello(3) 1455
os16: fwrite(3) 1455
os16: getc(3) 1455

os16: getchar(3) 1455
os16: getenv(3) 1455
os16: getopt(3) 1456
os16: getpwent(3) 1459
os16: getpwnam(3) 1460
os16: getpwuid(3) 1461
os16: gets(3) 1461
os16: heap(3) 1461
os16: heap_clear(3) 1462
os16: heap_min(3) 1462
os16: input_line(3) 1462
os16: isatty(3) 1463
os16: labs(3) 1463
os16: ldiv(3) 1463
os16: major(3) 1463
os16: makedev(3) 1463
os16: malloc(3) 1464
os16: memccpy(3) 1465
os16: memchr(3) 1465
os16: memcmp(3) 1465
os16: memcpy(3) 1466
os16: memmove(3) 1466
os16: memset(3) 1466
os16: minor(3) 1467
os16: namep(3) 1467
os16: offsetof(3) 1468
os16: opendir(3) 1468
os16: perror(3) 1468
os16: printf(3) 1469
os16: process_info(3) 1471
os16: putc(3) 1472
os16: putchar(3) 1472
os16: putenv(3) 1472
os16: puts(3) 1473
os16: qsort(3) 1473
os16: rand(3) 1473
os16: readdir(3) 1474
os16: realloc(3) 1474
os16: rewind(3) 1475
os16: rewinddir(3) 1475
os16: scanf(3) 1475
os16: seg_d(3) 1479
os16: seg_i(3) 1479
os16: setbuf(3) 1479
os16: setenv(3) 1479
os16: setpwent(3) 1480
os16: setvbuf(3) 1480
os16: snprintf(3) 1480
os16: sp(3) 1480
os16: sprintf(3) 1480
os16: srand(3) 1480

os16: ss(3) 1481
os16: sscanf(3) 1481
os16: stdio(3) 1481
os16: strcat(3) 1482
os16: strchr(3) 1482
os16: strcmp(3) 1483
os16: strcoll(3) 1483
os16: strcpy(3) 1483
os16: strcspn(3) 1484
os16: strdup(3) 1484
os16: strerror(3) 1484
os16: strlen(3) 1484
os16: strncat(3) 1485
os16: strncmp(3) 1485
os16: strncpy(3) 1485
os16: strpbrk(3) 1485
os16: strrchr(3) 1485
os16: strspn(3) 1486
os16: strstr(3) 1486
os16: strtok(3) 1486
os16: strtol(3) 1488
os16: strtoul(3) 1489
os16: strxfrm(3) 1489
os16: ttyname(3) 1489
os16: unsetenv(3) 1490
os16: vfprintf(3) 1490
os16: vfscanf(3) 1490
os16: vprintf(3) 1490
os16: vscanf(3) 1491
os16: vsnprintf(3) 1492
os16: vsprintf(3) 1492
os16: vsscanf(3) 1492

abort() 1435 abs() 1435 access() 1434 asctime() 1439 atexit() 1435 atoi() 1436 atol() 1436 basename() 1437 bp() 1438 clearerr() 1437 closedir() 1438 creat() 1438 cs() 1438 ctime() 1439 dirname() 1437 div() 1440 ds() 1438 endpwent() 1459 errfn 1441 errln 1441 errno 1441 errset() 1441 es() 1438 execl() 1445 execle() 1445 execlp() 1445 execv() 1445 execvp() 1445 exit() 1435 fclose() 1446 feof() 1447 ferror() 1447 fflush() 1448 fgetc() 1448 fgetpos() 1448 fgets() 1449 fileno() 1450 fopen() 1451 fprintf() 1469 fputc() 1452 fputs() 1452 fread() 1453 free() 1464 freopen() 1451 fscanf() 1475 fseek() 1453 fseeko() 1453 fsetpos() 1448 ftell() 1454 ftello() 1454 fwrite() 1455 getc() 1448 getchar() 1448 getenv() 1455 getopt() 1456 getpwent() 1459 getpwnam() 1460 getpwuid() 1460 gets() 1449 gmtime() 1439 heap_clear() 1461 heap_min() 1461 input_line() 1462 isatty() 1463 labs() 1435 ldiv() 1440 localtime() 1439 major() 1463 makedev() 1463 malloc() 1464 memccpy() 1465 memchr() 1465 memcmp() 1465 memcpy() 1466 memmove() 1466 memset() 1466 minor() 1463 mktime() 1439 namep() 1467 offsetof() 1468 opendir() 1468 perror() 1468 printf() 1469 process_info() 1471

putc() 1452 putchar() 1452 putenv() 1472 puts() 1452 qsort() 1473 rand() 1473 readdir() 1474 realloc() 1464 rewind() 1475 rewinddir() 1475 scanf() 1475 seg_d() 1479 seg_i() 1479 setbuf() 1479 setenv() 1479 setpwent() 1459 setvbuf() 1479 snprintf() 1469 sp() 1438 sprintf() 1469 srand() 1473 ss() 1438 sscanf() 1475 stdio.h 1481 strcat() 1482 strchr() 1482 strcmp() 1483 strcoll() 1483 strcpy() 1483 strcspn() 1486 strdup() 1484 strerror() 1484 strlen() 1484 strncat() 1482 strncmp() 1483 strncpy() 1483 strpbrk() 1485 strrchr() 1482 strspn() 1486 strstr() 1486 strtok() 1486 strtol() 1488 strtoul() 1488 strxfrm() 1489 ttyname() 1489 unsetenv() 1479 vfprintf() 1490 vfscanf() 1491 vprintf() 1490 vscanf() 1491 vsnprintf() 1490 vsprintf() 1490 vsscanf() 1491

os16: access(3)

NOME

'**access**' - verifica dei permessi di accesso dell'utente

SINTASSI

```
#include <unistd.h>
int access (const char *path, int mode);
```

DESCRIZIONE

La funzione ***access()*** verifica lo stato di accessibilità del file indicato nella stringa ***path***, secondo i permessi stabiliti con il parametro ***mode***.

L'argomento corrispondente al parametro ***mode*** può assumere un valore corrispondente alla macro-variabile ***F_OK***, per verificare semplicemente l'esistenza del file specificato; altrimenti, può avere un valore composto dalla combinazione (con l'operatore OR binario) di ***R_OK***, ***W_OK*** e ***X_OK***, per verificare, rispettivamente, l'accessibilità in lettura, in scrittura e in esecuzione. Queste macro-variabili sono dichiarate nel file '`unistd.h`'.

VALORE RESTITUITO

Valore	Significato
0	I permessi di accesso richiesti sono tutti disponibili.
-1	I permessi non sono tutti disponibili, oppure si è verificato un errore di altro genere, da chiarire analizzando la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
ENFILE	Troppi file aperti nel sistema.
ENOENT	File non trovato.
EACCES	Permesso negato.

DIFETTI

Questa realizzazione della funzione ***access()*** determina l'accessibilità a un file attraverso le informazioni che può trarre autonomamente, senza usare una chiamata di sistema. Pertanto, si tratta di una valutazione presunta e non reale.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/access.c`' [i161.17.2]

VEDERE ANCHE

stat(2) [u0.36].

os16: abort(3)

NOME

'**abort**' - conclusione anormale del processo

SINTASSI

```
#include <stdlib.h>
void abort (void);
```

DESCRIZIONE

La funzione ***abort()*** verifica lo stato di configurazione del segnale '**SIGABRT**' e, se risulta bloccato, lo sblocca, quindi invia questo segnale per il processo in corso. Ciò provoca la conclusione del processo, secondo la modalità prevista per tale segnale, a meno che il segnale sia stato ridiretto a una funzione, nel qual caso, dopo l'invio del segnale, potrebbe esserci anche una ripresa del controllo da parte della funzione ***abort()***. Tuttavia, se così fosse, il segnale '**SIGABRT**' verrebbe poi riconfigurato alla sua impostazione normale e verrebbe inviato nuovamente lo stesso segnale per provocare la conclusione del processo. Pertanto, la funzione ***abort()*** non restituisce il controllo.

Va comunque osservato che os16 non è in grado di associare una funzione a un segnale, pertanto, i segnali possono solo avere una gestione predefinita, o al massimo risultare bloccati.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/abort.c`' [i161.10.2]

VEDERE ANCHE

signal(2) [u0.34].

os16: abs(3)

NOME

'**abs**', '**labs**' - valore assoluto di un numero intero

SINTASSI

```
#include <stdlib.h>
int abs (int j);
long int labs (long int j);
```

DESCRIZIONE

Le funzioni '...**abs()**' restituiscono il valore assoluto del loro argomento. Si distinguono per tipo di intero e, nel caso di os16, non essendo disponibile il tipo '**long long int**', si limitano a ***abs()*** e ***labs()***.

VALORE RESTITUITO

Il valore assoluto del numero intero fornito come argomento.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/abs.c`' [i161.10.3]

'`lib/stdlib/labs.c`' [i161.10.12]

VEDERE ANCHE

div(3) [u0.15], *ldiv(3)* [u0.15], *rand(3)* [u0.85].

os16: atexit(3)

NOME

'**atexit**', '**exit**' - gestione della chiusura dei processi

SINTASSI

```
#include <stdlib.h>
typedef void (*atexit_t) (void);
int  atexit (atexit_t function);
void exit   (int status);
```

DESCRIZIONE

La funzione ***exit()*** conclude il processo in corso, avvalendosi della chiamata di sistema *_exit(2)* [u0.2], ma prima di farlo, scandisce un array contenente un elenco di funzioni, da eseguire prima di tale chiamata finale. Questo array viene popolato eventualmente con l'aiuto della funzione ***atexit()***, sapendo che l'ultima funzione di chiusura aggiunta, è la prima a dover essere eseguita alla conclusione.

La funzione ***atexit()*** riceve come argomento il puntatore a una funzione che non prevede argomenti e non restituisce alcunché. Per facilitare la dichiarazione del prototipo e, di conseguenza, dell'array usato per accumulare tali puntatori, il file di intestazione '`stdlib.h`' di os16 dichiara un tipo speciale, non standard, denominato '**`atexit_t`**', definito come:

```
typedef void (*atexit_t) (void);
```

Si possono annotare un massimo di ***ATEXIT_MAX*** funzioni da eseguire prima della conclusione di un processo. Tale macrovariabile è definita nel file '`limits.h`'.

VALORE RESTITUITO

Solo la funzione ***atexit()*** restituisce un valore, perché ***exit()*** non può nemmeno restituire il controllo.

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore, dovuto all'esaurimento dello spazio nell'array usato per accumulare le funzioni di chiusura.

FILE SORGENTI

'`lib/limits.h`' [i161.1.8]

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/atexit.c`' [i161.10.5]

'`lib/stdlib/exit.c`' [i161.10.10]

VEDERE ANCHE

_exit(2) [u0.2], *_Exit(2)* [u0.2].

os16: atoi(3)

«

NOME

'**`atoi`**', '**`atol`**' - conversione da stringa a numero intero

SINTASSI

```
#include <stdlib.h>
int atoi (const char *string);
long int atol (const char *string);
```

DESCRIZIONE

Le funzioni '**`ato...()`**' convertono una stringa, fornita come argomento, in un numero intero. La conversione avviene escludendo gli spazi iniziali, considerando eventualmente un segno («+» o «−») e poi soltanto i caratteri che rappresentano cifre numeriche. La scansione della stringa e l'interpretazione del valore numerico contenuto terminano quando si incontra un carattere diverso dalle cifre numeriche.

VALORE RESTITUITO

Il valore numerico ottenuto dall'interpretazione della stringa.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/atoi.c`' [i161.10.6]

'`lib/stdlib/atol.c`' [i161.10.7]

VEDERE ANCHE

strtol(3) [u0.121], *strtoul(3)* [u0.121].

os16: atol(3)

«

Vedere *atoi(3)* [u0.5].

os16: basename(3)

«

NOME

'**`basename`**', '**`dirname`**' - elaborazione dei componenti di un percorso

SINTASSI

```
#include <libgen.h>
char *basename (char *path);
char *dirname (char *path);
```

DESCRIZIONE

Le funzioni ***basename()*** e ***dirname()***, restituiscono un percorso, estratto da quello fornito come argomento (***path***). Per la precisione, ***basename()*** restituisce l'ultimo componente del percorso, mentre ***dirname()*** restituisce ciò che precede l'ultimo componente. Valgono gli esempi seguenti:

Contenuto originale di ***path***	Risultato prodotto da '**`dirname(`*path*`)`**'	Risultato prodotto da '**`basename(`*path*`)`**'
`"/usr/bin/"`	`"/usr"`	`"bin"`
`"/usr/bin"`	`"/usr"`	`"bin"`
`"/usr"`	`"/"`	`"usr"`
`"usr"`	`"."`	`"usr"`
`"/"`	`"/"`	`"/"`
`"."`	`"."`	`"."`
`".."`	`".."`	`".."`

È importante considerare che le due funzioni alterano il contenuto di ***path***, in modo da isolare i componenti che servono.

VALORE RESTITUITO

Le due funzioni restituiscono il puntatore alla stringa contenente il risultato dell'elaborazione, trattandosi di una porzione della stringa già usata come argomento della chiamata e modificata per l'occasione. Non è previsto il manifestarsi di alcun errore.

FILE SORGENTI

'`lib/libgen.h`' [u0.6]

'`lib/libgen/basename.c`' [i161.6.1]

'`lib/libgen/dirname.c`' [i161.6.2]

os16: bp(3)

«

Vedere *cs(3)* [u0.12].

os16: clearerr(3)

«

NOME

'**`clearerr`**' - azzeramento degli indicatori di errore e di fine file di un certo flusso di file

SINTASSI

```
#include <stdio.h>
void clearerr (FILE *fp);
```

DESCRIZIONE

La funzione ***clearerr()*** azzera gli indicatori di errore e di fine file, del flusso di file indicato come argomento.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/clearerr.c`' [i161.9.2]

VEDERE ANCHE

feof(3) [u0.28], *ferror(3)* [u0.29], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: closedir(3)

NOME

'**closedir**' - chiusura di una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
int closedir (DIR *dp);
```

DESCRIZIONE

La funzione ***closedir()*** chiude la directory rappresentata da ***dp***.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
EBADF	La directory rappresentata da ***dp***, non è valida.

FILE SORGENTI

'lib/sys/types.h' [u0.14]

'lib/dirent.h' [u0.2]

'lib/dirent/DIR.c' [i161.2.1]

'lib/dirent/closedir.c' [i161.2.2]

VEDERE ANCHE

close(2) [u0.7], *opendir(3)* [u0.76], *readdir(3)* [u0.86], *rewinddir(3)* [u0.89].

os16: creat(3)

NOME

'**creat**' - creazione di un file puro e semplice

SINTASSI

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat (const char *path, mode_t mode);
```

DESCRIZIONE

La funzione ***creat()*** equivale esattamente all'uso della funzione ***open()***, con le opzioni '**O_WRONLY|O_CREAT|O_TRUNC**':

```
open (path, O_WRONLY|O_CREAT|O_TRUNC, mode)
```

Per ogni altra informazione, si veda la pagina di manuale *open(2)* [u0.28].

FILE SORGENTI

'lib/sys/types.h' [u0.14]

'lib/sys/stat.h' [u0.13]

'lib/fcntl.h' [u0.4]

'lib/fcntl/creat.c' [i161.4.1]

VEDERE ANCHE

chmod(2) [u0.4], *chown(2)* [u0.5], *close(2)* [u0.7], *dup(2)* [u0.8], *fcntl(2)* [u0.13], *link(2)* [u0.24], *mknod(2)* [u0.26], *mount(2)* [u0.27], *open(2)* [u0.28] *read(2)* [u0.29], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42], *write(2)* [u0.44], *fopen(3)* [u0.35].

os16: cs(3)

NOME

'**bp**', '**cs**', '**ds**', '**es**', '**sp**', '**ss**' - stato dei registri della CPU

SINTASSI

```
#include <sys/os16.h>
unsigned int cs (void);
unsigned int ds (void);
unsigned int ss (void);
unsigned int es (void);
unsigned int sp (void);
unsigned int bp (void);
```

DESCRIZIONE

Le funzioni elencate nel quadro sintattico, sono in realtà delle macroistruzioni, chiamanti funzioni con nomi analoghi, ma preceduti da un trattino basso (***_bp()***, ***_cs()***, ***_ds()***, ***_es()***, ***_sp()***, ***_ss()***), per interrogare lo stato di alcuni registri della CPU a fini diagnostici. I registri interessati sono quelli con lo stesso nome della macroistruzione usata per interrogarli.

FILE SORGENTI

'lib/sys/os16.h' [u0.12]

'lib/sys/os16/_cs.s' [i161.12.2]

'lib/sys/os16/_ds.s' [i161.12.3]

'lib/sys/os16/_ss.s' [i161.12.8]

'lib/sys/os16/_es.s' [i161.12.4]

'lib/sys/os16/_sp.s' [i161.12.7]

'lib/sys/os16/_bp.s' [i161.12.1]

VEDERE ANCHE

seg_i(3) [u0.91].

seg_d(3) [u0.91].

os16: ctime(3)

NOME

'**asctime**', '**ctime**', '**gmtime**', '**localtime**', '**mktime**' - conversione di informazioni data-orario

SINTASSI

```
#include <time.h>
char      *asctime   (const struct tm *timeptr);
char      *ctime     (const time_t *timer);
struct tm *gmtime    (const time_t *timer);
struct tm *localtime (const time_t *timer);
time_t     mktime    (const struct tm *timeptr);
```

DESCRIZIONE

Queste funzioni hanno in comune il compito di convertire delle informazioni data-orario, da un formato a un altro, eventualmente anche testuale. Una data e un orario possono essere rappresentati con il tipo '**time_t**', nel qual caso si tratta del numero di secondi trascorsi dall'ora zero del 1 gennaio 1970; in alternativa potrebbe essere rappresentata in una variabile strutturata di tipo '**struct tm**', dichiarato nel file 'time.h':

```
struct tm {
    int tm_sec;     // secondi
    int tm_min;     // minuti
    int tm_hour;    // ore
    int tm_mday;    // giorno del mese
    int tm_mon;     // mese, da 1 a 12
    int tm_year;    // anno
    int tm_wday;    // giorno della settimana,
                    // da 0 (domenica) a 6
    int tm_yday;    // giorno dell'anno
    int tm_isdst;   // informazioni sull'ora estiva
};
```

In alcuni casi, la conversione dovrebbe tenere conto della configurazione locale, ovvero del fuso orario ed eventualmente del cambiamento di orario nel periodo estivo. os16 non considera alcunché e gestisce il tempo in un modo assoluto, senza nozione della convenzione locale.

La funzione ***asctime()*** converte quanto contenuto in una variabile strutturata di tipo '**`struct tm`**' in una stringa che descrive la data e l'ora in inglese. La stringa in questione è allocata staticamente e viene sovrascritta se la funzione viene usata più volte.

La funzione ***ctime()*** è in realtà soltanto una macroistruzione, la quale, complessivamente, converte il tempo indicato come quantità di secondi, restituendo il puntatore a una stringa che descrive la data attuale, tenendo conto della configurazione locale. La stringa in questione utilizza la stessa memoria statica usata per ***asctime()***; inoltre, dato che os16 non distingue tra ora locale e tempo universale, la funzione non esegue alcuna conversione temporale.

La funzione ***gmtime()*** converte il tempo espresso in secondi in una forma suddivisa, secondo il tipo '**`struct tm`**'. La funzione restituisce quindi il puntatore a una variabile strutturata di tipo '**`struct tm`**', la quale però è dichiarata in modo statico, internamente alla funzione, e viene sovrascritta nelle chiamate successive della stessa.

La funzione ***localtime()*** converte una data espressa in secondi, in una data suddivisa in campi ('**`struct tm`**'), tenendo conto (ma non succede con os16) della configurazione locale.

La funzione ***mktime()*** converte una data contenute in una variabile strutturata di tipo '**`struct tm`**' nella quantità di secondi corrispondente.

VALORE RESTITUITO

Le funzioni che restituiscono un puntatore, se incontrano un errore, restituiscono il puntatore nullo, '**`NULL`**'. Nel caso particolare di ***mktime()***, se il valore restituito è pari a `-1`, si tratta di un errore.

FILE SORGENTI

'`lib/time.h`' [u0.16]

'`lib/time/asctime.c`' [i161.16.1]

'`lib/time/gmtime.c`' [i161.16.3]

'`lib/time/mktime.c`' [i161.16.4]

VEDERE ANCHE

date(1) [u0.8], *clock(2)* [u0.6], *time(2)* [u0.39].

os16: dirname(3)

«

Vedere *basename(3)* [u0.7].

os16: div(3)

«

NOME

'**`div`**', '**`ldiv`**' - calcolo del quoziente e del resto di una divisione intera

SINTASSI

```
#include <stdlib.h>
div_t div (int numer, int denom);
ldiv_t ldiv (long int numer, long int denom);
```

DESCRIZIONE

Le funzioni '...**`div()`**' calcolano la divisione tra numeratore e denominatore, forniti come argomenti della chiamata, restituendo un risultato, composto di divisione intera e resto, in una variabile strutturata.

I tipi '**`div_t`**' e '**`ldiv_t`**', sono dichiarati nel file '`stdlib.h`' nel modo seguente:


```
typedef struct {
    int      quot;
    int      rem;
} div_t;
//
typedef struct {
    long int quot;
    long int rem;
} ldiv_t;
```

I membri ***quot*** contengono il quoziente, ovvero il risultato intero; i membri ***rem*** contengono il resto della divisione.

VALORE RESTITUITO

Il risultato della divisione, strutturato in quoziente e resto.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/div.c`' [i161.10.8]

'`lib/stdlib/ldiv.c`' [i161.10.13]

VEDERE ANCHE

abs(3) [u0.3].

os16: ds(3)

«

Vedere *cs(3)* [u0.12].

os16: endpwent(3)

«

Vedere *getpwent(3)* [u0.53].

os16: errno(3)

«

NOME

'**`errno`**' - numero dell'ultimo errore riportato

SINTASSI

```
#include <errno.h>
```

DESCRIZIONE

Attraverso l'inclusione del file '`errno.h`', si ottiene la dichiarazione della variabile ***errno***. In pratica, per os16 viene dichiarata così:

```
extern int errno;
```

Per annotare un errore, si assegna un valore numerico a questa variabile. Il valore numerico in questione rappresenta sinteticamente la descrizione dell'errore; pertanto, si utilizzano per questo delle macro-variabili, dichiarate tutte nel file '`errno.h`'. Nell'esempio seguente si annota l'errore ***ENOMEM***, corrispondente alla descrizione «Not enough space», ovvero «spazio insufficiente»:

```
errno = ENOMEM;
```

Dal momento che in questo modo viene comunque a mancare un riferimento al sorgente e alla posizione in cui l'errore si è manifestato, la libreria di os16 aggiunge la macroistruzione ***errset()***, la quale però non fa parte dello standard. Si usa così:

```
errset (ENOMEM);
```

La macroistruzione ***errset()*** aggiorna la variabile ***errln*** e l'array ***errfn[]***, rispettivamente con il numero della riga di codice in cui si è manifestato il problema e il nome della funzione che lo contiene. Con queste informazioni, la funzione *perror(3)* [u0.77] può visualizzare più dati.

Pertanto, nel codice di os16, si usa sempre la macroistruzione ***errset()***, invece di assegnare semplicemente un valore alla variabile ***errno***.

ERRORI

Gli errori previsti dalla libreria di os16 sono riassunti dalla tabella successiva. La prima parte contiene gli errori definiti dallo standard POSIX, ma solo alcuni di questi vengono usati effettivamente nella libreria, data la limitatezza di os16.

Valore di ***errno***	Definizione
E2BIG	Argument list too long.
EACCES	Permission denied.
EADDRINUSE	Address in use.
EADDRNOTAVAIL	Address not available.
EAFNOSUPPORT	Address family not supported.

Valore di ***errno***	Definizione
EAGAIN	Resource unavailable, try again.
EALREADY	Connection already in progress.
EBADF	Bad file descriptor.
EBADMSG	Bad message.
EBUSY	Device or resource busy.

Valore di ***errno***	Definizione
ECANCELED	Operation canceled.
ECHILD	No child processes.
ECONNABORTED	Connection aborted.
ECONNREFUSED	Connection refused.
ECONNRESET	Connection reset.

Valore di ***errno***	Definizione
EDEADLK	Resource deadlock would occur.
EDESTADDRREQ	Destination address required.
EDOM	Mathematics argument out of domain of function.
EDQUOT	Reserved.
EEXIST	File exists.

Valore di ***errno***	Definizione
EFAULT	Bad address.
EFBIG	File too large.
EHOSTUNREACH	Host is unreachable.
EIDRM	Identifier removed.
EILSEQ	Illegal byte sequence.

Valore di ***errno***	Definizione
EINPROGRESS	Operation in progress.
EINTR	Interrupted function.
EINVAL	Invalid argument.
EIO	I/O error.
EISCONN	Socket is connected.

Valore di ***errno***	Definizione
EISDIR	Is a directory.
ELOOP	Too many levels of symbolic links.
EMFILE	Too many open files.
EMLINK	Too many links.
EMSGSIZE	Message too large.

Valore di ***errno***	Definizione
EMULTIHOP	Reserved.
ENAMETOOLONG	Filename too long.
ENETDOWN	Network is down.
ENETRESET	Connection aborted by network.
ENETUNREACH	Network unreachable.

Valore di ***errno***	Definizione
ENFILE	Too many files open in system.
ENOBUFS	No buffer space available.
ENODATA	No message is available on the stream head read queue.
ENODEV	No such device.
ENOENT	No such file or directory.

Valore di ***errno***	Definizione
ENOEXEC	Executable file format error.
ENOLCK	No locks available.
ENOLINK	Reserved.
ENOMEM	Not enough space.
ENOMSG	No message of the desired type.

Valore di ***errno***	Definizione
ENOPROTOOPT	Protocol not available.
ENOSPC	No space left on device.
ENOSR	No stream resources.
ENOSTR	Not a stream.
ENOSYS	Function not supported.

Valore di ***errno***	Definizione
ENOTCONN	The socket is not connected.
ENOTDIR	Not a directory.
ENOTEMPTY	Directory not empty.
ENOTSOCK	Not a socket.
ENOTSUP	Not supported.

Valore di *errno*	Definizione
ENOTTY	Inappropriate I/O control operation.
ENXIO	No such device or address.
EOPNOTSUPP	Operation not supported on socket.
EOVERFLOW	Value too large to be stored in data type.
EPERM	Operation not permitted.

Valore di *errno*	Definizione
EPIPE	Broken pipe.
EPROTO	Protocol error.
EPROTONOSUPPORT	Protocol not supported.
EPROTOTYPE	Protocol wrong type for socket.
ERANGE	Result too large.

Valore di *errno*	Definizione
EROFS	Read-only file system.
ESPIPE	Invalid seek.
ESRCH	No such process.
ESTALE	Reserved.
ETIME	Stream ioctl() timeout.

Valore di *errno*	Definizione
ETIMEDOUT	Connection timed out.
ETXTBSY	Text file busy.
EWOULDBLOCK	Operation would block (may be the same as EAGAIN).
EXDEV	Cross-device link.

La tabella successiva raccoglie le definizioni degli errori aggiuntivi, specifici di os16.

Valore di *errno*	Definizione
EUNKNOWN	Unknown error.
E_FILE_TYPE	File type not compatible.
E_ROOT_INODE_NOT_CACHED	The root directory inode is not cached.
E_CANNOT_READ_SUPERBLOCK	Cannot read super block.
E_MAP_INODE_TOO_BIG	Map inode too big.

Valore di *errno*	Definizione
E_MAP_ZONE_TOO_BIG	Map zone too big.
E_DATA_ZONE_TOO_BIG	Data zone too big.
E_CANNOT_FIND_ROOT_DEVICE	Cannot find root device.
E_CANNOT_FIND_ROOT_INODE	Cannot find root inode.
E_FILE_TYPE_UNSUPPORTED	File type unsupported.

Valore di *errno*	Definizione
E_ENV_TOO_BIG	Environment too big.
E_LIMIT	Exceeded implementation limits.
E_NOT_MOUNTED	Not mounted.
E_NOT_IMPLEMENTED	Not implemented.

FILE SORGENTI

'`lib/errno.h`' [u0.3]

'`lib/errno/errno.c`' [i161.3.1]

VEDERE ANCHE

perror(3) [u0.77], *strerror(3)* [u0.111].

os16: es(3)

Vedere *cs(3)* [u0.12].

os16: exec(3)

NOME

'**`execl`**', '**`execle`**', '**`execlp`**', '**`execv`**', '**`execvp`**' - esecuzione di un file

SINTASSI

```
#include <unistd.h>
extern char **environ;
int execl  (const char *path, const char *arg, ...);
int execle (const char *path, const char *arg, ...);
int execlp (const char *path, const char *arg, ...);
int execv  (const char *path, char *const argv[]);
int execvp (const char *path, char *const argv[]);
```

DESCRIZIONE

Le funzioni '**`exec`**...**`()`**' rimpiazzano il processo chiamante con un altro processo, ottenuto dal caricamento di un file eseguibile in memoria. Tutte queste funzioni si avvalgono, direttamente o indirettamente di *execve(2)* [u0.10].

Il primo argomento delle funzioni descritte qui è il percorso, rappresentato dalla stringa ***path***, di un file da eseguire.

Le funzioni '**`execl`**...**`()`**', dopo il percorso del file eseguibile, richiedono l'indicazione di una quantità variabile di argomenti, a cominciare da ***arg***, costituiti da stringhe, tenendo conto che dopo l'ultimo di questi argomenti, occorre fornire il puntatore nullo, '**`NULL`**', per chiarire che questi sono terminati. L'argomento corrispondente al parametro ***arg*** deve essere una stringa contenente il nome del file da avviare, mentre gli argomenti successivi sono gli argomenti da passare al programma stesso.

Rispetto al gruppo di funzioni '**`execl`**...**`()`**', la funzione ***execle()***, dopo il puntatore nullo che conclude la sequenza di argomenti per il programma da avviare, si attende una sequenza di altre stringhe, anche questa conclusa da un ultimo e definitivo puntatore nullo. Questa ulteriore sequenza di stringhe va a costituire l'ambiente del processo da avviare, pertanto il contenuto di tali stringhe deve essere del tipo '***nome=valore***'.

Le funzioni '**`execv`**...**`()`**' hanno come ultimo argomento il puntatore a un array di stringhe, dove ***argv[0]*** deve essere il nome del file da avviare, mentre da ***argv[1]*** in poi, si tratta degli argomenti da passare al programma. Anche in questo caso, per riconoscere l'ultimo elemento di questo array, gli si deve assegnare il puntatore nullo.

Le funzioni ***execlp()*** e ***execvp()***, se ricevono solo il nome del file da eseguire, senza altre indicazioni del percorso, cercano questo file tra i percorsi indicati nella variabile di ambiente ***PATH***, ammesso che sia dichiarata per il processo in corso.

Tutte le funzioni qui descritte, a eccezione di ***execle()***, trasmettono al nuovo processo lo stesso ambiente (le stesse variabili di ambiente) del processo chiamante.

VALORE RESTITUITO

Queste funzioni, se hanno successo nel loro compito, non possono restituire alcunché, dato che in quel momento, il processo chiamante viene rimpiazzato da quello del file che viene eseguito. Pertanto, queste funzioni possono restituire soltanto un valore che rappresenta un errore, ovvero `-1`, aggiornando anche la variabile ***errno*** di conseguenza.

ERRORI

Valore di ***errno***	Significato
`E2BIG`	Ci sono troppi argomenti.
`ENOMEM`	Memoria insufficiente.
`ENOENT`	Il file richiesto non esiste.
`EACCES`	Il file non può essere avviato per la mancanza dei permessi di acceso necessari.
`ENOEXEC`	Il file non può essere un file eseguibile, perché non ne ha le caratteristiche.
`EIO`	Errore di input-output.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/execl.c`’ [i161.17.9]

‘`lib/unistd/execle.c`’ [i161.17.10]

‘`lib/unistd/execlp.c`’ [i161.17.11]

‘`lib/unistd/execv.c`’ [i161.17.12]

‘`lib/unistd/execvp.c`’ [i161.17.14]

‘`lib/unistd/execve.c`’ [i161.17.13]

VEDERE ANCHE

execve(2) [u0.10], *fork(2)* [u0.14], *environ(7)* [u0.1].

os16: execl(3)

Vedere *exec(3)* [u0.20].

os16: execle(3)

Vedere *exec(3)* [u0.20].

os16: execlp(3)

Vedere *exec(3)* [u0.20].

os16: execv(3)

Vedere *exec(3)* [u0.20].

os16: execvp(3)

Vedere *exec(3)* [u0.20].

os16: exit(3)

Vedere *atexit(3)* [u0.4].

os16: fclose(3)

NOME

‘**`fclose`**’ - chiusura di un flusso di file

SINTASSI

```
#include <stdio.h>
int fclose (FILE *fp);
```


DESCRIZIONE

La funzione ***fclose()*** chiude il flusso di file specificato tramite il puntatore ***fp***. Questa realizzazione particolare di os16, si limita a richiamare la funzione ***close()***, con l’indicazione del descrittore di file corrispondente al flusso.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
‘`EOF`’	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso da chiudere, non è valido.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/fclose.c`’ [i161.9.3]

VEDERE ANCHE

close(2) [u0.7], *fopen(3)* [u0.35], *stdio(3)* [u0.103].

os16: feof(3)

NOME

‘**`feof`**’ - verifica dello stato dell’indicatore di fine file

SINTASSI

```
#include <stdio.h>
int feof (FILE *fp);
```

DESCRIZIONE

La funzione ***feof()*** restituisce il valore dell’indicatore di fine file, riferito al flusso di file rappresentato da ***fp***.

VALORE RESTITUITO

Valore	Significato
diverso da zero	Significa che l’indicatore di fine file è impostato.
0	Significa che l’indicatore di fine file non è impostato.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/feof.c`’ [i161.9.4]

VEDERE ANCHE

clearerr(3) [u0.9], *ferror(3)* [u0.29], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: ferror(3)

NOME

‘**`ferror`**’ - verifica dello stato dell’indicatore di errore

SINTASSI

```
#include <stdio.h>
int ferror (FILE *fp);
```

DESCRIZIONE

La funzione ***ferror()*** restituisce il valore dell’indicatore di errore, riferito al flusso di file rappresentato da ***fp***.

VALORE RESTITUITO

Valore	Significato
diverso da zero	Significa che l’indicatore di errore è impostato.
0	Significa che l’indicatore di errore non è impostato.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/ferror.c`’ [i161.9.5]

VEDERE ANCHE

clearerr(3) [u0.9], *feof(3)* [u0.28], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: fflush(3)

NOME

'**fflush**' - fissaggio dei dati ancora sospesi nella memoria tampone

SINTASSI

```
#include <stdio.h>
int fflush (FILE *fp);
```

DESCRIZIONE

La funzione ***fflush()*** di os16, non fa alcunché, dato che non è prevista alcuna gestione della memoria tampone per i flussi di file.

VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/fflush.c' [i161.9.6]

VEDERE ANCHE

fclose(3) [u0.27], *fopen(3)* [u0.35].

os16: fgetc(3)

NOME

'**fgetc**', '**getc**', '**getchar**' - lettura di un carattere da un flusso di file

SINTASSI

```
#include <stdio.h>
int fgetc   (FILE *fp);
int getc    (FILE *fp);
int getchar (void);
```

DESCRIZIONE

Le funzioni ***fgetc()*** e ***getc()*** sono equivalenti e leggono il carattere successivo dal flusso di file rappresentato da ***fp***. La funzione ***getchar()*** esegue la lettura di un carattere, ma dallo standard input.

VALORE RESTITUITO

In caso di successo, il carattere letto viene restituito in forma di intero positivo (il carattere viene inteso inizialmente senza segno, quindi viene trasformato in un intero, il quale rappresenta così un valore positivo). Se la lettura non può avere luogo, la funzione restituisce '**EOF**', corrispondente a un valore negativo.

ERRORI

La variabile ***errno*** potrebbe risultare aggiornata nel caso la funzione restituisca '**EOF**'. Ma per saperlo, occorre azzerare la variabile ***errno*** prima della chiamata di ***fgetc()***.

Valore di ***errno***	Significato
EBADF	Il descrittore di file associato al flusso da cui leggere un carattere, non è valido.
EINVAL	Il descrittore di file associato al flusso da cui leggere un carattere, non consente un accesso in lettura.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/fgetc.c' [i161.9.7]

'lib/stdio/getchar.c' [i161.9.24]

VEDERE ANCHE

fgets(3) [u0.33], *gets(3)* [u0.33].

os16: fgetpos(3)

NOME

'**fgetpos**', '**fsetpos**' - lettura e impostazione della posizione corrente di un flusso di file

SINTASSI

```
#include <stdio.h>
int fgetpos (FILE *restrict fp, fpos_t *restrict pos);
int fsetpos (FILE *restrict fp, fpos_t *restrict pos);
```

DESCRIZIONE

Le funzioni ***fgetpos()*** e ***fsetpos()***, rispettivamente, leggono o impostano la posizione corrente di un flusso di file.

Per os16, il tipo '**fpos_t**' è dichiarato nel file 'stdio.h' come equivalente al tipo '**off_t**' (file 'sys/types.h'); tuttavia, la modifica di una variabile di tipo '**fpos_t**' va fatta utilizzando una funzione apposita, perché lo standard consente che possa trattarsi anche di una variabile strutturata, i cui membri non sono noti.

L'uso della funzione ***fgetpos()*** comporta una modifica dell'informazione contenuta all'interno di ****pos***, mentre la funzione ***fsetpos()*** usa il valore contenuto in ****pos*** per cambiare la posizione corrente del flusso di file. In pratica, si può usare ***fsetpos()*** solo dopo aver salvato una certa posizione con l'aiuto di ***fgetpos()***.

Si comprende che il valore restituito dalle funzioni è solo un indice del successo o meno dell'operazione, dato che l'informazione sulla posizione viene ottenuta dalla modifica di una variabile di cui si fornisce il puntatore negli argomenti.

VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	Il descrittore di file associato al flusso, non è valido.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/fgetpos.c' [i161.9.8]

'lib/stdio/fsetpos.c' [i161.9.20]

VEDERE ANCHE

fseek(3) [u0.43], *ftell(3)* [u0.46], *rewind(3)* [u0.88].

os16: fgets(3)

NOME

'**fgets**', '**gets**' - lettura di una stringa da un flusso di file

SINTASSI

```
#include <stdio.h>
char *fgets (char *restrict string, int n,
             FILE *restrict fp);
char *gets  (char *string);
```

DESCRIZIONE

La funzione ***fgets()*** legge una «riga» dal flusso di file ***fp***, purché non più lunga di ***n***−1 caratteri, collocandola a partire da ciò a cui punta ***stringa***. La lettura termina al raggiungimento del carattere

'**\n**' (*new line*), oppure alla fine del file, oppure a ***n***−1 caratteri. In ogni caso, viene aggiunto al termine, il codice nullo di terminazione di stringa: '**\0**'.

La funzione ***gets()***, in modo analogo a ***fgets()***, legge una riga dallo standard input, ma senza poter porre un limite massimo alla lunghezza della lettura.

VALORE RESTITUITO

In caso di successo, viene restituito il puntatore alla stringa contenente la riga letta, ovvero restituiscono ***string***. In caso di errore, o comunque in caso di una lettura nulla, si ottiene '**NULL**'. La variabile ***errno*** viene aggiornata solo se si presenta un errore di accesso al file, mentre una lettura nulla, perché il flusso si è concluso, non comporta tale aggiornamento.

ERRORI

La variabile ***errno*** potrebbe risultare aggiornata nel caso le funzioni restituiscano '**NULL**'. Ma per saperlo, occorre azzerare la variabile ***errno*** prima della chiamata di queste.

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso da cui leggere un carattere, non è valido.
`EINVAL`	Il descrittore di file associato al flusso da cui leggere un carattere, non consente un accesso in lettura.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/fgets.c`' [i161.9.9]

'`lib/stdio/gets.c`' [i161.9.25]

VEDERE ANCHE

fgetc(3) [u0.31], *getc(3)* [u0.31].

os16: fileno(3)

NOME

'**fileno**' - traduzione di un flusso di file nel numero di descrittore corrispondente

SINTASSI

```
#include <stdio.h>
int fileno (FILE *fp);
```

DESCRIZIONE

La funzione ***fileno()*** traduce il flusso di file, rappresentato da ***fp***, nel numero del descrittore corrispondente. Tuttavia, il risultato è valido solo se il flusso di file specificato è aperto effettivamente.

VALORE RESTITUITO

Se ***fp*** punta effettivamente a un flusso di file aperto, il valore restituito corrisponde al numero di descrittore del file stesso; diversamente, si potrebbe ottenere un numero privo di senso. Se come argomento si indica il puntatore nullo, si ottiene un errore, rappresentato dal valore `-1`.

ERRORI

Valore di ***errno***	Significato
`EBADF`	È stato richiesto di risolvere il puntatore nullo.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/fileno.c`' [i161.9.10]

VEDERE ANCHE

clearerr(3) [u0.9], *feof(3)* [u0.28], *ferror(3)* [u0.29], *stdio(3)* [u0.103].

os16: fopen(3)

NOME

'**fopen**', '**freopen**' - apertura di un flusso di file

SINTASSI

```
#include <stdio.h>
FILE *fopen   (const char *path, const char *mode);
FILE *freopen (const char *restrict path,
               const char *restrict mode,
               FILE *restrict fp);
```

DESCRIZIONE

La funzione ***fopen()*** apre il file indicato nella stringa a cui punta ***path***, secondo le modalità di accesso contenute in ***mode***, associandovi un flusso di file. In modo analogo agisce anche la funzione ***freopen()***, la quale però, prima, chiude il flusso ***fp***.

La modalità di accesso al file si specifica attraverso una stringa, come sintetizzato dalla tabella successiva.

mode	Significato
`"r"` `"rb"`	Si richiede un accesso in lettura, di un file già esistente. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.

`"r+"`
`"r+b"`
`"rb+"`
	Si richiede un accesso in lettura e scrittura, di un file già esistente. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.

`"w"` `"wb"`	Si richiede un accesso in scrittura, di un file che viene troncato se esiste oppure viene creato. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.

`"w+"`
`"w+b"`
`"wb+"`
	Si richiede un accesso in lettura e scrittura, di un file che viene troncato se esiste oppure viene creato. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.

`"a"` `"ab"`	Si richiede la creazione o il troncamento di un file, con accesso in aggiunta. L'indice interno per l'accesso ai dati viene posizionato alla fine del file, prima di ogni operazione di scrittura.

`"a+"`
`"a+b"`
`"ab+"`
	Si richiede la creazione o il troncamento di un file, con accesso in lettura e scrittura. L'indice interno per l'accesso ai dati viene posizionato alla fine del file, prima di ogni operazione di scrittura.

VALORE RESTITUITO

Se l'operazione si conclude con successo, viene restituito il puntatore a ciò che rappresenta il flusso di file aperto. Se però si ottiene un puntatore nullo ('**NULL**'), si è verificato un errore che può essere interpretato dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	È stato fornito un argomento non valido.
`EPERM`	Operazione non consentita.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/FILE.c`’ [i161.9.1]

‘`lib/stdio/fopen.c`’ [i161.9.11]

‘`lib/stdio/freopen.c`’ [i161.9.16]

VEDERE ANCHE

open(2) [u0.28], *fclose(3)* [u0.27], *stdio(3)* [u0.103].

os16: fprintf(3)

Vedere *printf(3)* [u0.78].

os16: fputc(3)

NOME

‘**`fputc`**’, ‘**`putc`**’, ‘**`putchar`**’ - emissione di un carattere attraverso un flusso di file

SINTASSI

```
#include <stdio.h>
int fputc   (int c, FILE *fp);
int putc    (int c, FILE *fp);
int putchar (int c);
```

DESCRIZIONE

Le funzioni ***fputc()*** e ***putc()*** sono equivalenti e scrivono il carattere ***c*** nel flusso di file rappresentato da ***fp***. La funzione ***putchar()*** esegue la scrittura di un carattere, ma nello standard output.

VALORE RESTITUITO

In caso di successo, il carattere scritto viene restituito in forma di intero positivo (il carattere viene inteso inizialmente senza segno, quindi viene trasformato in un intero, il quale rappresenta così un valore positivo). Se la scrittura non può avere luogo, la funzione restituisce ‘**`EOF`**’, corrispondente a un valore negativo.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso presso cui scrivere un carattere, non è valido.
`EINVAL`	Il descrittore di file associato al flusso presso cui scrivere un carattere, non consente un accesso in scrittura.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/fputc.c`’ [i161.9.13]

VEDERE ANCHE

fputs(3) [u0.38], *puts(3)* [u0.38].

os16: fputs(3)

NOME

‘**`fputs`**’, ‘**`puts`**’ - scrittura di una stringa attraverso un flusso di file

SINTASSI

```
#include <stdio.h>
int fputs (const char *restrict string, FILE *restrict fp);
int puts  (const char *string);
```

DESCRIZIONE

La funzione ***fputs()*** scrive una stringa nel flusso di file ***fp***, ma senza il carattere nullo di terminazione; la funzione ***puts()*** scrive una stringa, aggiungendo anche il codice di terminazione ‘**`\n`**’, attraverso lo standard output.

VALORE RESTITUITO

Valore	Significato
≥ 0	Rappresenta il successo dell’operazione.
‘**`EOF`**’	Indica il verificarsi di un errore, il quale può essere interpretato eventualmente leggendo la variabile ***errno***.

ERRORI

La variabile ***errno*** potrebbe risultare aggiornata nel caso le funzioni restituiscano ‘**`NULL`**’. Ma per saperlo, occorre azzerare la variabile ***errno*** prima della chiamata di queste.

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso verso cui si deve scrivere, non è valido.
`EINVAL`	Il descrittore di file associato al flusso verso cui si deve scrivere, non consente un accesso in scrittura.

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/fputs.c`’ [i161.9.14]

‘`lib/stdio/puts.c`’ [i161.9.28]

VEDERE ANCHE

fputc(3) [u0.37], *putc(3)* [u0.37], *putchar(3)* [u0.37].

os16: fread(3)

NOME

‘**`fread`**’ - lettura di dati da un flusso di file

SINTASSI

```
#include <stdio.h>
size_t fread (void *restrict buffer, size_t size,
              size_t nmemb, FILE *restrict fp);
```

DESCRIZIONE

La funzione ***fread()*** legge ***size***×***nmemb*** byte dal flusso di file ***fp***, trascrivendoli in memoria a partire dall’indirizzo a cui punta ***buffer***.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letta, diviso la dimensione del blocco ***nmemb*** (byte/***nmemb***). Se il valore ottenuto è inferiore a quello richiesto, occorre verificare eventualmente se ciò deriva dalla conclusione del file o da un errore, con l’aiuto di *feof(3)* [u0.28] e di *ferror(3)* [u0.29].

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/fread.c`’ [i161.9.15]

VEDERE ANCHE

read(2) [u0.29], *write(2)* [u0.44], *feof(3)* [u0.28], *ferror(3)* [u0.29], *fwrite(3)* [u0.48].

os16: free(3)

Vedere *malloc(3)* [u0.66].

os16: freopen(3)

Vedere *fopen(3)* [u0.35].

os16: fscanf(3)

Vedere *scanf(3)* [u0.90].

os16: fseek(3)

NOME

‘**`fseek`**’, ‘**`fseeko`**’ - riposizionamento dell’indice di accesso di un flusso di file

SINTASSI

```
#include <stdio.h>
int fseek  (FILE *fp, long int offset, int whence);
int fseeko (FILE *fp, off_t offset, int whence);
```

DESCRIZIONE

Le funzioni ***fseek()*** e ***fseeko()*** cambiano l'indice della posizione interna a un flusso di file, specificato dal parametro ***fp***. L'indice viene collocato secondo lo scostamento rappresentato da ***offset***, rispetto al riferimento costituito dal parametro ***whence***. Il parametro ***whence*** può assumere solo tre valori, come descritto nello schema successivo.

Valore di ***whence***	Significato
`SEEK_SET`	lo scostamento si riferisce all'inizio del file.
`SEEK_CUR`	lo scostamento si riferisce alla posizione che ha già l'indice interno al file.
`SEEK_END`	lo scostamento si riferisce alla fine del file.

La differenza tra le due funzioni sta solo nel tipo del parametro ***offset***, il quale, da '**`long int`**' passa a '**`off_t`**'.

VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso, non è valido.
`EINVAL`	Gli argomenti non sono validi, come succede se la combinazione di scostamento e riferimento non è ammissibile (per esempio uno scostamento negativo, quando il riferimento è l'inizio del file).

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/fseek.c`' [i161.9.18]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.46], *rewind(3)* [u0.88].

os16: fseeko(3)

«

Vedere *fseek(3)* [u0.43].

os16: fsetpos(3)

«

Vedere *fgetpos(3)* [u0.32].

os16: ftell(3)

«

NOME

'**`ftell`**', '**`ftello`**' - interrogazione dell'indice di accesso relativo a un flusso di file

SINTASSI

```
#include <stdio.h>
long int ftell  (FILE *fp);
off_t    ftello (FILE *fp);
```

DESCRIZIONE

Le funzioni ***ftell()*** e ***ftello()*** restituiscono il valore dell'indice interno di accesso al file specificato in forma di flusso, con il parametro ***fp***. La differenza tra le due funzioni consiste nel tipo restituito, il quale, nel primo caso è '**`long int`**', mentre nel secondo è '**`off_t`**'. L'indice ottenuto è riferito all'inizio del file.

VALORE RESTITUITO

Valore	Significato
≥ 0	Rappresenta l'indice interno al file.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso, non è valido.
`EINVAL`	Il flusso di file specificato non è valido.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/ftell.c`' [i161.9.21]

'`lib/stdio/ftello.c`' [i161.9.22]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.43], *rewind(3)* [u0.88].

os16: ftello(3)

«

Vedere *ftell(3)* [u0.46].

os16: fwrite(3)

«

NOME

'**`fwrite`**' - scrittura attraverso un flusso di file

SINTASSI

```
#include <stdio.h>
size_t fwrite (const void *restrict buffer, size_t size,
               size_t nmemb, FILE *restrict fp);
```

DESCRIZIONE

La funzione ***fwrite()*** scrive ***size***×***nmemb*** byte nel flusso di file ***fp***, traendoli dalla memoria, a partire dall'indirizzo a cui punta ***buffer***.

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritta, diviso la dimensione del blocco rappresentato da ***nmemb*** (byte/***nmemb***). Se il valore ottenuto è inferiore a quello richiesto, si tratta presumibilmente di un errore, ma per accertarsene conviene usare *ferror(3)* [u0.29].

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/fwrite.c`' [i161.9.23]

VEDERE ANCHE

read(2) [u0.29], *write(2)* [u0.44], *feof(3)* [u0.28], *ferror(3)* [u0.29], *fread(3)* [u0.39].

os16: getc(3)

«

Vedere *fgetc(3)* [u0.31].

os16: getchar(3)

«

Vedere *fgetc(3)* [u0.31].

os16: getenv(3)

«

NOME

'**`getenv`**' - lettura del valore di una variabile di ambiente

SINTASSI

```
#include <stdlib.h>
char *getenv (const char *name);
```

DESCRIZIONE

La funzione ***getenv()*** richiede come argomento una stringa contenente il nome di una variabile di ambiente, per poter restituire la stringa che rappresenta il contenuto di tale variabile.

VALORE RESTITUITO

Il puntatore alla stringa con il contenuto della variabile di ambiente richiesta, oppure il puntatore nullo (‘**NULL**’), se la variabile in questione non esiste.

FILE SORGENTI

‘`lib/stdlib.h`’ [u0.10]

‘`applic/crt0.s`’ [i162.1.9]

‘`lib/stdlib/environment.c`’ [i161.10.9]

‘`lib/stdlib/getenv.c`’ [i161.10.11]

VEDERE ANCHE

environ(7) [u0.1], *putenv(3)* [u0.82], *setenv(3)* [u0.94], *unsetenv(3)* [u0.94].

os16: getopt(3)

NOME

‘**getopt**’ - scansione delle opzioni della riga di comando

SINTASSI

```
#include <unistd.h>
extern *char optarg;
extern int   optind;
extern int   opterr;
extern int   optopt;
int getopt (int argc, char *const argv[],
            const char *optstring);
```

DESCRIZIONE

La funzione ***getopt()*** riceve, come primi due argomenti, gli stessi parametri ***argc*** e ***argv[]***, che sono già della funzione ***main()*** del programma in cui ***getopt()*** si usa. In altri termini, ***getopt()*** deve conoscere la quantità degli argomenti usati per l’avvio del programma e deve poterli scandire. L’ultimo argomento di ***getopt()*** è una stringa contenente l’elenco delle lettere delle opzioni che ci si attende di trovare nella scansione delle stringhe dell’array ***argv[]***, con altre sigle eventuali per sapere se tali opzioni sono singole o si attendono un proprio argomento.

Per poter usare la funzione ***getopt()*** proficuamente, è necessario che la sintassi di utilizzo del programma del quale si vuole scandire la riga di comando, sia uniforme con l’uso comune:

```
programma [-x[ argomento]]... [argomento]...
```

Pertanto, dopo il nome del programma possono esserci delle opzioni, riconoscibili perché composte da una sola lettera, preceduta da un trattino. Tali opzioni potrebbero richiedere un proprio argomento. Dopo le opzioni e i relativi argomenti, ci possono essere altri argomenti, al di fuori della competenza di ***getopt()***. Vale anche la considerazione che più opzioni, prive di argomento, possono essere unite assieme in un’unica parola, con un solo trattino iniziale.

La funzione ***getopt()*** si avvale di variabili pubbliche, di cui occorre conoscere lo scopo.

La variabile ***optind*** viene usata da ***getopt()*** come indice per scandire l’array ***argv[]***. Quando con gli utilizzi successivi di ***optarg()*** si determina che è stata completata la scansione delle opzioni (in quanto ***optarg()*** restituisce il valore -1), la variabile ***optind*** diventa utile per conoscere qual è il prossimo elemento di ***argv[]*** da prendere in considerazione, trattandosi del primo argomento della riga di comando che non è un’opzione.

La variabile ***opterr*** serve per configurare il comportamento di ***getopt()***. Questa variabile contiene inizialmente il valore 1. Quando ***getopt()*** incontra un’opzione per la quale si richiede un argomento, il quale risulta però mancante, se la variabile ***opterr*** risulta avere un valore diverso da zero, visualizza un messaggio di errore attraverso lo standard error. Pertanto, per evitare tale visualizzazione, è sufficiente assegnare preventivamente il valore zero alla variabile ***opterr***.

Quando un’opzione individuata da ***getopt()*** risulta errata per qualche ragione (perché non prevista o perché si attende un argomento che invece non c’è), la variabile ***optopt*** riceve il valore (tradotto da carattere senza segno a intero) della lettera corrispondente a quell’opzione. Pertanto, in tal modo è possibile conoscere cosa ha provocato il problema.

Il puntatore ***optarg*** viene modificato quando ***getopt()*** incontra un’opzione che chiede un argomento. In tal caso, ***optarg*** viene modificato in modo da puntare alla stringa che rappresenta tale argomento.

La compilazione della stringa corrispondente a ***optstring*** deve avvenire secondo una sintassi precisa:

```
[:][x[:]]...
```

La stringa ***optstring*** può iniziare con un simbolo di due punti, quindi seguono le lettere che rappresentano le opzioni possibili, tenendo conto che quelle per cui si attende un argomento devono anche essere seguite da due punti. Per esempio, ‘**ab:cd:**’ significa che ci può essere un’opzione ‘**-a**’, un’opzione ‘**-b**’ seguita da un argomento, un’opzione ‘**-c**’ e un’opzione ‘**-d**’ seguita da un argomento.

Per comprendere l’uso della funzione ***getopt()*** si propone una versione ultraridotta di *kill(1)* [u0.10], dove si ammette solo l’invio dei segnali SIGTERM e SIGQUIT.

```
#include <sys/os16.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
//------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
  int          signal = SIGTERM;
  int          pid;
  int          a;              // Index inside arguments.
  int          opt;
  extern char *optarg;
  extern int   optopt;
  //
  while ((opt = getopt (argc, argv, ":ls:")) != -1)
    {
      switch (opt)
        {
          case 'l':
              printf ("TERM ");
              printf ("KILL ");
              printf ("\n");
              return (0);
              break;
          case 's':
              if (strcmp (optarg, "KILL") == 0)
                {
                  signal = SIGKILL;
```

```
            }
          else if (strcmp (optarg, "TERM") == 0)
            {
              signal = SIGTERM;
            }
          break;
        case '?':
          fprintf (stderr, "Unknown option -%c.\n",
                   optopt);
          return (1);
          break;
        case ':':
          fprintf (stderr,
                   "Missing argument for option "
                   "-%c\n",
                   optopt);
          return (1);
          break;
        default:
          fprintf (stderr,
                   "Getopt problem: unknown option "
                   "%c\n", opt);
          return (1);
      }
  }
//
// Scan other command line arguments.
//
for (a = optind; a < argc; a++)
  {
    pid = atoi (argv[a]);
    if (pid > 0)
      {
        if (kill (pid, signal) < 0)
          {
            perror (argv[a]);
          }
      }
  }
return (0);
}
```

Come si vede nell'esempio, la funzione ***getopt()*** viene chiamata sempre nello stesso modo, all'interno di un ciclo iterativo.

Alla prima chiamata della funzione, questa esamina il primo argomento della riga di comando, verificando se si tratta di un'opzione o meno. Se si tratta di un'opzione, benché possa essere errata per qualche ragione, restituisce un carattere (convertito a intero), il quale può corrispondere alla lettera dell'opzione se questa è valida, oppure a un simbolo differente in caso di problemi. Nelle chiamate successive, ***getopt()*** considera di volta in volta gli argomenti successivi della riga di comando, fino a quando si accorge che non ci sono più opzioni e restituisce semplicemente il valore `-1`.

Durante la scansione delle opzioni, se ***getopt()*** restituisce il carattere '**`?`**', significa che ha incontrato un'opzione errata: potrebbe trattarsi di un'opzione non prevista, oppure di un'opzione che attende un argomento che non c'è. Tuttavia, la stringa ***optstring*** potrebbe iniziare opportunamente con il simbolo di due punti, così come si vede nell'esempio. In tal caso, se ***getopt()*** incontra un'opzione errata in quanto mancante di un'opzione necessaria, invece di restituire '**`?`**', restituisce '**`:`**', così da poter distinguere il tipo di errore.

È il caso di osservare che le chiamate successive di ***getopt()*** fanno progredire la scansione della riga di comando e generalmente non c'è bisogno di tornare indietro per ripeterla. Tuttavia, nel caso lo si volesse, basterebbe reinizializzare la variabile ***optind*** a uno (il primo argomento della riga di comando).

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/getopt.c`' [i161.17.20]

NOME

'**`getpwent`**', '**`setpwent`**', '**`endpwent`**' - accesso alle voci del file '`/etc/passwd`'

SINTASSI

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwent (void);
void           setpwent (void);
void           endpwent (void);
```

DESCRIZIONE

La funzione ***getpwent()*** restituisce il puntatore a una variabile strutturata, di tipo '**`struct passwd`**', come definito nel file '`pwd.h`', in cui si possono trovare le stesse informazioni contenute nelle voci (righe) del file '`/etc/passwd`', separate in campi. La prima volta, nella variabile struttura a cui punta la funzione si ottiene il contenuto della prima voce, ovvero del primo utente dell'elenco; nelle chiamate successive si ottengono le altre.

Si utilizza la funzione ***setpwent()*** per ripartire dalla prima voce del file '`/etc/passwd`'; si utilizza invece la funzione ***endpwent()*** per chiudere il file '`/etc/passwd`' quando non serve più.

Il tipo '**`struct passwd`**' è definito nel file '`pwd.h`' nel modo seguente:

```
struct passwd {
    char  *pw_name;
    char  *pw_passwd;
    uid_t  pw_uid;
    gid_t  pw_gid;
    char  *pw_gecos;
    char  *pw_dir;
    char  *pw_shell;
};
```

La sequenza dei campi della struttura corrisponde a quella contenuta nel file '`/etc/passwd`'.

VALORE RESTITUITO

La funzione ***getpwent()*** restituisce il puntatore a una variabile strutturata di tipo '**`struct passwd`**', se l'operazione ha avuto successo. Se la scansione del file '`/etc/passwd`' ha raggiunto il termine, oppure se si è verificato un errore, restituisce invece il valore '**`NULL`**'. Per poter distinguere tra la conclusione del file o il verificarsi di un errore, prima della chiamata della funzione occorre azzerare il valore della variabile ***errno***, verificando successivamente se ha acquisito un valore differente.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

‘`lib/sys/types.h`’ [u0.14]

‘`lib/pwd.h`’ [u0.7]

‘`lib/pwd/pwent.c`’ [i161.7.1]

VEDERE ANCHE

getpwnam(3) [u0.54], *getpwuid(3)* [u0.54], *passwd(5)* [u0.3].

os16: getpwnam(3)

«

NOME

‘**`getpwnam`**’, ‘**`getpwuid`**’ - selezione di una voce dal file ‘`/etc/passwd`’

SINTASSI

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwnam (const char *name);
struct passwd *getpwuid (uid_t uid);
```

DESCRIZIONE

La funzione ***getpwnam()*** restituisce il puntatore a una variabile strutturata, di tipo ‘**`struct passwd`**’, come definito nel file ‘`pwd.h`’, contenente le informazioni sull’utenza specificata per nome, dal ‘`/etc/passwd`’. La funzione ***getpwuid()*** si comporta in modo analogo, individuando però l’utenza da selezionare in base al numero UID.

Il tipo ‘**`struct passwd`**’ è definito nel file ‘`pwd.h`’ nel modo seguente:

```
struct passwd {
    char  *pw_name;
    char  *pw_passwd;
    uid_t  pw_uid;
    gid_t  pw_gid;
    char  *pw_gecos;
    char  *pw_dir;
    char  *pw_shell;
};
```

La sequenza dei campi della struttura corrisponde a quella contenuta nel file ‘`/etc/passwd`’.

VALORE RESTITUITO

Le funzioni ***getpwnam()*** e ***getpwuid()*** restituiscono il puntatore a una variabile strutturata di tipo ‘**`struct passwd`**’, se l’operazione ha avuto successo. Se il nome o il numero dell’utente non si trovano nel file ‘`/etc/passwd`’, oppure se si presenta un errore, il valore restituito è ‘**`NULL`**’. Per poter distinguere tra una voce non trovata o il verificarsi di un errore di accesso al file ‘`/etc/passwd`’, prima della chiamata della funzione occorre azzerare il valore della variabile ***errno***, verificando successivamente se ha acquisito un valore differente.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell’ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell’ambito del processo in corso.

FILE SORGENTI

‘`lib/sys/types.h`’ [u0.14]

‘`lib/pwd.h`’ [u0.7]

‘`lib/pwd/pwent.c`’ [i161.7.1]

VEDERE ANCHE

getpwent(3) [u0.53], *setpwent(3)* [u0.53], *endpwent(3)* [u0.53], *passwd(5)* [u0.3].

os16: getpwuid(3)

Vedere *getpwnam(3)* [u0.54].

«

os16: gets(3)

Vedere *fgets(3)* [u0.33].

«

os16: heap(3)

«

NOME

‘**`heap_clear`**’, ‘**`heap_min`**’ - verifica dello spazio disponibile per la pila dei dati

SINTASSI

```
#include <sys/os16.h>
void heap_clear   (void);
int  heap_min     (void);
```

DESCRIZIONE

Le funzioni ***heap_clear()*** e ***heap_min()*** servono per poter conoscere, in un certo momento, lo spazio di memoria disponibile per la pila dei dati, durante il funzionamento del processo elaborativo.

La funzione ***heap_clear()*** sovrascrive la memoria tra la fine della memoria utilizzata per le variabili non inizializzate (BSS) e la parte superiore della pila dei dati. In altri termini, sovrascrive la parte di memoria disponibile per la pila dei dati, che in quel momento non è utilizzata. Vengono scritte sequenze di bit a uno.

La funzione ***heap_min()***, da usare successivamente a ***heap_clear()***, anche più avanti nell’esecuzione del processo, scandisce questa memoria e verifica, empiricamente, il livello minimo di memoria rimasto libero per la pila, in base all’utilizzo che se ne è fatto fino a quel punto. In pratica, serve a verificare se il programma da cui ha origine il processo ha uno spazio sufficiente per la pila dei dati o se ci sia il rischio di sovrapposizione con le altre aree dei dati.

VALORE RESTITUITO

La funzione ***heap_min()*** restituisce la quantità di byte di memoria continua, presumibilmente non ancora utilizzata dalla pila dei dati, che separa la pila stessa dalle altre aree di dati.

FILE SORGENTI

‘`lib/sys/os16.h`’ [u0.12]

‘`lib/sys/os16/heap_clear.c`’ [i161.12.9]

‘`lib/sys/os16/heap_min.c`’ [i161.12.10]

VEDERE ANCHE

cs(3) [u0.12], *ds(3)* [u0.12], *es(3)* [u0.12], *ss(3)* [u0.12], *bp(3)* [u0.12], *sp(3)* [u0.12].

os16: heap_clear(3)

Vedere *heap(3)* [u0.57].

os16: heap_min(3)

Vedere *heap(3)* [u0.57].

os16: input_line(3)

NOME

‘**`input_line`**’ - riga di comando

SINTASSI

```
#include <sys/os16.h>
void input_line (char *line, char *prompt, size_t size,
                 int type);
```

DESCRIZIONE

La funzione ***input_line()*** consente di inserire un’informazione da tastiera, interpretando in modo adeguato i codici usati per cancellare. Si tratta dell’unico mezzo corretto di inserimento di un dato da tastiera, per os16, il quale non dispone di una gestione completa dei terminali.

Il parametro ***line*** è il puntatore a un’area di memoria, da modificare con l’inserimento che si intende fare; questa area di memoria deve essere in grado di contenere tanti byte quanto indicato con il parametro ***size***. Il parametro ***prompt*** indica una stringa da usare come invito, a sinistra della riga da inserire. Il parametro ***type*** serve a specificare il tipo di visualizzazione sullo schermo di ciò che si inserisce. Si utilizzano della macro-variabili dichiarate nel file ‘`sys/os16.h`’:

Macro-variabile	Descrizione
`INPUT_LINE_ECHO`	Produce un comportamento «normale», per cui ciò che viene digitato è rappresentato conformemente sullo schermo del terminale.
`INPUT_LINE_HIDDEN`	Fa sì che quanto digitato non appaia: si usa per esempio per l’inserimento di una parola d’ordine.
`INPUT_LINE_STARS`	Fa sì che per ogni carattere digitato appaia sullo schermo un asterisco: si usa per esempio per l’inserimento di una parola d’ordine, quando si vuole agevolare l’utente.

La funzione conclude il suo funzionamento quando si preme [*Invio*].

VALORE RESTITUITO

La funzione non restituisce alcunché, ma ciò che viene digitato è disponibile nella memoria tampone rappresentata dal puntatore ***line***, da intendere come stringa terminata correttamente.

FILE SORGENTI

‘`lib/sys/os16.h`’ [u0.12]

‘`lib/sys/os16/input_line.c`’ [i161.12.11]

VEDERE ANCHE

shell(1) [u0.19].

login(1) [u0.12].

os16: isatty(3)

NOME

‘**`isatty`**’ - verifica che un certo descrittore di file si riferisca a un terminale

SINTASSI

```
#include <unistd.h>
int isatty (int fdn);
```

DESCRIZIONE

La funzione ***isatty()*** verifica se il descrittore di file specificato con il parametro ***fdn*** si riferisce a un dispositivo di terminale.

VALORE RESTITUITO

Valore	Significato
1	Si tratta effettivamente del file di dispositivo di un terminale.
0	Il descrittore di file non riguarda un terminale, oppure si è verificato un errore; in ogni caso la variabile ***errno*** viene aggiornata.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il descrittore di file non si riferisce a un terminale.
`EBADF`	Il descrittore di file indicato non è valido.

FILE SORGENTI

‘`lib/unistd.h`’ [u0.17]

‘`lib/unistd/isatty.c`’ [i161.17.25]

VEDERE ANCHE

stat(2) [u0.36], *ttyname(3)* [u0.124].

os16: labs(3)

Vedere *abs(3)* [u0.3].

os16: ldiv(3)

Vedere *div(3)* [u0.15].

os16: major(3)

Vedere *makedev(3)* [u0.65].

os16: makedev(3)

NOME

‘**`makedev`**’, ‘**`major`**’, ‘**`minor`**’ - gestione dei numeri di dispositivo

SINTASSI

```
#include <sys/types.h>
dev_t makedev (int major, int minor);
int   major   (dev_t device);
int   minor   (dev_t device);
```

DESCRIZIONE

La funzione ***makedev()*** restituisce il numero di dispositivo complessivo, partendo dal numero primario (***major***) e dal numero secondario (***minor***), presi separatamente.

Le funzioni ***major()*** e ***minor()***, rispettivamente, restituiscono il numero primario o il numero secondario, partendo da un numero di dispositivo completo.

Si tratta di funzioni non previste dallo standard, ma ugualmente diffuse.

FILE SORGENTI

‘`lib/sys/types.h`’ [u0.14]

‘`lib/sys/types/makedev.c`’ [i161.14.2]

‘`lib/sys/types/major.c`’ [i161.14.1]

‘`lib/sys/types/minor.c`’ [i161.14.3]

os16: malloc(3)

NOME

‘**`malloc`**’, ‘**`free`**’, ‘**`realloc`**’ - allocazione e rilascio dinamico di memoria

SINTASSI

```
#include <stdlib.h>
void *malloc (size_t size);
void free (void *address);
void *realloc (void *address, size_t size);
```

DESCRIZIONE

Le funzioni ‘…**`alloc()`**’ e ***free()*** consentono di allocare, riallocare e liberare delle aree di memoria, in modo dinamico.

La funzione ***malloc()*** (*memory allocation*) si usa per richiedere l’allocazione di una dimensione di almeno ***size*** byte di memoria. Se l’allocazione avviene con successo, la funzione restituisce il puntatore generico di tale area allocata.

Quando un’area di memoria allocata precedentemente non serve più, va liberata espressamente con l’ausilio della funzione ***free()***, la quale richiede come argomento il puntatore generico all’inizio di tale area. Naturalmente, si può liberare la memoria una volta sola e un’area di memoria liberata non può più essere raggiunta.

Quando un’area di memoria già allocata richiede una modifica nella sua estensione, si può usare la funzione ***realloc()***, la quale necessita di conoscere il puntatore precedente e la nuova estensione. La funzione restituisce un nuovo puntatore, il quale potrebbe eventualmente, ma non necessariamente, coincidere con quello dell’area originale.

Se le funzioni ***malloc()*** e ***realloc()*** falliscono nel loro intento, restituiscono un puntatore nullo.

VALORE RESTITUITO

Le funzioni ***malloc()*** e ***realloc()*** restituiscono il puntatore generico all’area di memoria allocata; se falliscono, restituiscono invece un puntatore nullo.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente.

DIFETTI

L’allocazione dinamica di memoria, della libreria di os16, utilizza un metodo rudimentale, basato su un array statico che viene allocato completamente se nella compilazione si utilizzano queste funzioni. Questo array, denominato ***_alloc_memory[]***, viene utilizzato come area per l’allocazione della memoria, con l’ausilio di altre due variabili allo scopo di tenere traccia della mappa di allocazione. In pratica, la memoria che si può gestire in questo modo è molto poca, ma soprattutto, i processi che ne fanno uso, in realtà, la allocano subito tutta.

FILE SORGENTI

‘`lib/limits.h`’ [i161.1.8]

‘`lib/stdlib.h`’ [u0.10]

‘`lib/stdlib/alloc.c`’ [i161.10.4]

os16: memccpy(3)

NOME

‘**`memccpy`**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memccpy (void *restrict dst,
               const void *restrict org,
               int c, size_t n);
```

DESCRIZIONE

La funzione ***memccpy()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***, fermandosi se si incontra il carattere ***c***, il quale viene copiato regolarmente, fermo restando il limite massimo di ***n*** byte.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

VALORE RESTITUITO

Nel caso in cui la copia sia avvenuta con successo, fino a incontrare il carattere ***c***, la funzione restituisce il puntatore al carattere successivo a ***c***, nell’area di memoria di destinazione. Se invece tale carattere non viene trovato nei primi ***n*** byte, restituisce il puntatore nullo ‘**`NULL`**’. La variabile ***errno*** non viene modificata.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memccpy.c`’ [i161.11.1]

VEDERE ANCHE

memcpy(3) [u0.70], *memmove(3)* [u0.71], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: memchr(3)

NOME

‘**`memchr`**’ - scansione della memoria alla ricerca di un carattere

SINTASSI

```
#include <string.h>
void *memchr (const void *memory, int c, size_t n);
```

DESCRIZIONE

La funzione ***memchr()*** scandisce l’area di memoria a cui punta ***memory***, fino a un massimo di ***n*** byte, alla ricerca del carattere ***c***.

VALORE RESTITUITO

Se la funzione trova il carattere, restituisce il puntatore al carattere trovato, altrimenti restituisce il puntatore nullo ‘**`NULL`**’.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memchr.c`’ [i161.11.2]

VEDERE ANCHE

strchr(3) [u0.105], *strrchr(3)* [u0.105], *strpbrk(3)* [u0.116].

os16: memcmp(3)

NOME

‘**`memcmp`**’ - confronto di due aree di memoria

SINTASSI

```
#include <string.h>
int memcmp (const void *memory1, const void *memory2,
            size_t n);
```

DESCRIZIONE

La funzione ***memcmp()*** confronta i primi ***n*** byte di memoria delle aree che partono, rispettivamente, da ***memory1*** e da ***memory2***.

VALORE RESTITUITO

Valore	Esito del confronto.
-1	***memory1*** < ***memory2***
0	***memory1*** == ***memory2***
+1	***memory1*** > ***memory2***

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memcmp.c`’ [i161.11.3]

VEDERE ANCHE

strcmp(3) [u0.106], *strncmp(3)* [u0.106].

os16: memcpy(3)

NOME

‘**`memcpy`**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memcpy (void *restrict dst, const void *restrict org,
              size_t n);
```

DESCRIZIONE

La funzione ***memcpy()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

VALORE RESTITUITO

La funzione restituisce ***dst***.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memcpy.c`’ [i161.11.4]

VEDERE ANCHE

memccpy(3) [u0.67], *memmove(3)* [u0.71], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: memmove(3)

NOME

‘**`memmove`**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memmove (void *dst, const void *org, size_t n);
```

DESCRIZIONE

La funzione ***memmove()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***. A differenza di quanto fa ***memcpy()***, la funzione ***memmove()*** esegue la copia correttamente anche se le due aree di memoria sono sovrapposte.

VALORE RESTITUITO

La funzione restituisce ***dst***.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memmove.c`’ [i161.11.5]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: memset(3)

NOME

‘**`memset`**’ - scrittura della memoria con un byte sempre uguale

SINTASSI

```
#include <string.h>
void *memset (void *memory, int c, size_t n);
```

DESCRIZIONE

La funzione ***memset()*** scrive ***n*** byte, contenenti il valore di ***c***, ridotto a un carattere, a partire dal ciò a cui punta ***memory***.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/memset.c`’ [i161.11.6]

VEDERE ANCHE

memcpy(3) [u0.70].

os16: minor(3)

Vedere *makedev(3)* [u0.65].

os16: namep(3)

NOME

‘**`namep`**’ - ricerca del percorso di un programma utilizzando la variabile di ambiente ***PATH***

SINTASSI

```
#include <sys/os16.h>
int namep (const char *name, char *path, size_t size);
```

DESCRIZIONE

La funzione ***namep()*** trova il percorso di un programma, tenendo conto delle informazioni contenute nella variabile di ambiente ***PATH***.

Il parametro ***name*** rappresenta una stringa con il nome del comando da cercare nel file system; il parametro ***path*** deve essere il puntatore di un’area di memoria, da sovrascrivere con il percorso assoluto del programma da avviare, una volta trovato, con l’accortezza di far sì che risulti una stringa terminata correttamente; il parametro ***size*** specifica la dimensione massima che può avere la stringa ***path***.

Questa funzione viene utilizzata in particolare da ***execvp()***.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l’errore indicato dalla variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Uno degli argomenti non è valido.
`ENOENT`	La variabile di ambiente ***PATH*** non è dichiarata, oppure il comando richiesto non si trova nei percorsi previsti.
`ENAMETOOLONG`	Il percorso per l’avvio del programma è troppo lungo.

FILE SORGENTI

‘`lib/sys/os16.h`’ [u0.12]

‘`lib/sys/os16/namep.c`’ [i161.12.13]

VEDERE ANCHE

shell(1) [u0.19], *execvp(3)* [u0.20], *execlp(3)* [u0.20].

os16: offsetof(3)

NOME

'**offsetof**' - posizione di un membro di una struttura, dall'inizio della stessa

SINTASSI

```
#include <stddef.h>
size_t offsetof (type, member);
```

DESCRIZIONE

La macroistruzione ***offsetof()*** consente di determinare la collocazione relativa di un membro di una variabile strutturata, restituendo la quantità di byte che la struttura occupa prima dell'inizio del membro richiesto. Per ottenere questo risultato, il primo argomento deve essere il nome del tipo del membro cercato, mentre il secondo argomento deve essere il nome del membro stesso.

VALORE RESTITUITO

La macroistruzione restituisce lo scostamento del membro specificato, rispetto all'inizio della struttura a cui appartiene, espresso in byte.

FILE SORGENTI

'`lib/stddef.h`' [i161.1.14]

os16: opendir(3)

NOME

'**opendir**' - apertura di una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (const char *name);
```

DESCRIZIONE

La funzione ***opendir()*** apre la directory rappresentata da ***name***, posizionando l'indice interno per le operazioni di accesso alla prima voce della directory stessa.

VALORE RESTITUITO

La funzione restituisce il puntatore al flusso aperto; in caso di errore, restituisce '**NULL**' e aggiorna la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EINVAL	Il valore di ***name*** non è valido.
EMFILE	Troppi file aperti dal processo.
ENFILE	Troppi file aperti complessivamente nel sistema.
ENOTDIR	Il percorso indicato in ***name*** non corrisponde a una directory.

NOTE

La funzione ***opendir()*** attiva il bit *close-on-exec*, rappresentato dalla macro-variabile ***FD_CLOEXEC***, per il descrittore del file che rappresenta la directory. Ciò serve a garantire che la directory venga chiusa quando si utilizzano le funzioni '**exec**...**()**'.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/opendir.c`' [i161.2.3]

VEDERE ANCHE

open(2) [u0.28], *closedir(3)* [u0.10], *readdir(3)* [u0.86], *rewinddir(3)* [u0.89].

os16: perror(3)

NOME

'**perror**' - emissione di un messaggio di errore di sistema

SINTASSI

```
#include <stdio.h>
void perror (const char *string);
```

DESCRIZIONE

La funzione ***perror()*** legge il valore della variabile ***errno*** e, se questo è diverso da zero, emette attraverso lo standard output la stringa fornita come argomento, ammesso che non si tratti del puntatore nullo, quindi continua con l'emissione della descrizione dell'errore.

La funzione ***perror()*** di os16, nell'emettere il testo dell'errore, mostra anche il nome del file sorgente e il numero della riga in cui si è verificato. Ma questi dati sono validi soltanto se l'annotazione dell'errore è avvenuta, a suo tempo, con l'ausilio della funzione *errset(3)* [u0.18], la quale non è prevista dagli standard.

FILE SORGENTI

'`lib/errno.h`' [u0.3]

'`lib/stdio.h`' [u0.9]

'`lib/stdio/perror.c`' [i161.9.26]

VEDERE ANCHE

errno(3) [u0.18], *strerror(3)* [u0.111].

os16: printf(3)

NOME

'**printf**', '**fprintf**', '**sprintf**', '**snprintf**' - composizione dei dati per la visualizzazione

SINTASSI

```
#include <stdio.h>
int printf   (char *restrict format, ...);
int fprintf  (FILE *fp, char *restrict format, ...);
int sprintf  (char *restrict string,
              const char *restrict format, ...);
int snprintf (char *restrict string, size_t size,
              const char *restrict format, ...);
```

DESCRIZIONE

Le funzioni del gruppo '...**printf()**' hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e si collocano come argomenti finali, di tipo e quantità non noti nel prototipo delle funzioni. Per quantificare e qualificare questi argomenti aggiuntivi, la stringa a cui punta il parametro ***format***, deve contenere degli ***specificatori di conversione***, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta ***format***, la interpretano e determinano quali argomenti variabili sono presenti, quindi producono un'altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi. Si osservi l'esempio seguente:

```
printf ("Valore: %x %i %o\n", 123, 124, 125);
```

In questo modo si ottiene la visualizzazione, attraverso lo standard output, della stringa '**Valore: 7b 124 175**'. Infatti: '**%x**' è uno specificatore di conversione che richiede di interpretare il proprio parametro (in questo caso il primo) come intero normale e di rappresentarlo in esadecimale; '**%i**' legge un numero intero normale e lo rappresenta nella forma decimale consueta; '**%o**' legge un intero e lo mostra in ottale.

La funzione ***printf()*** emette il risultato della composizione attraverso lo standard output; la funzione ***fprintf()*** lo fa attraverso il flusso di file ***fp***; le funzioni ***sprintf()*** e ***snprintf()*** si limitano a scrivere il risultato a partire da ciò a cui punta ***string***, con la particolarità di ***snprintf()*** che si dà comunque un limite da non superare, per evitare che la scrittura vada a sovrascrivere altri dati in memoria.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di os16:

`%[`*simbolo*`][`*n_ampiezza*`][.`*n_precisione*`][hh|h|l|j|z|t]`*tipo*

La prima cosa da individuare in uno specificatore di conversione è il tipo di argomento che viene interpretato e, di conseguenza, il genere di rappresentazione che se ne vuole produrre. Il tipo viene espresso da una lettera alfabetica, alla fine dello specificatore di conversione.

Simbolo	Tipo di argomento	Conversione applicata
`%…d` `%…i`	`int`	Numero intero con segno da rappresentare in base dieci.
`%…u`	`unsigned int`	Numero intero senza segno da rappresentare in base dieci.
`%…o`	`unsigned int`	Numero intero senza segno da rappresentare in ottale (senza lo zero iniziale che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…x` `%…X`	`unsigned int`	Numero intero senza segno da rappresentare in esadecimale (senza il prefisso '**`0x`**' o '**`0X`**' che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…c`	`int`	Un carattere singolo, dopo la conversione in '**`unsigned char`**'.
`%…s`	`char *`	Una stringa.
`%%`		Questo specificatore si limita a produrre un carattere di percentuale ('**`%`**') che altrimenti non sarebbe rappresentabile.

Nel modello sintattico che descrive lo specificatore di conversione, si vede che subito dopo il segno di percentuale può apparire un simbolo (*flag*).

Simbolo	Corrispondenza
`%+…` `%+0`*ampiezza*`…`	Il segno «+» fa sì che i numeri con segno lo mostrino anche se è positivo. Può combinarsi con lo zero.
`%0`*ampiezza*`…` `%+0`*ampiezza*`…`	Lo zero fa sì che siano inseriti degli zeri a sinistra per allineare a destra il valore, nell'ambito dell'ampiezza specificata. Può combinarsi con il segno «+».
`%`*ampiezza*`…` `% `*ampiezza*`…`	In mancanza di uno zero iniziale, in presenza dell'indicazione dell'ampiezza, il valore viene allineato a destra usando degli spazi. È possibile esprimere esplicitamente l'intenzione di usare gli spazi mettendo proprio uno spazio, ma in generale non è richiesto. Se si mette lo spazio letteralmente, questo non è poi compatibile con lo zero, mentre le combinazioni con gli altri simboli sono ammissibili.
`%-`*ampiezza*`…` `%-+`*ampiezza*`…`	Il segno meno, usato quando la conversione prevede l'uso di una quantità fissa di caratteri con un valore che appare di norma allineato a destra, fa sì che il risultato sia allineato a sinistra. Il segno meno si può combinare il segno «+» e il cancelletto.

Subito prima della lettera che definisce il tipo di conversione, possono apparire una o due lettere che modificano la lunghezza del valore da interpretare (per lunghezza si intende qui la quantità di byte usati per rappresentarlo). Per esempio, '**`%…li`**' indica che la conversione riguarda un valore di tipo '**`long int`**'. Tra questi specificatori della lunghezza del dato in ingresso ce ne sono alcuni che indicano un rango inferiore a quello di '**`int`**', come per esempio '**`%…hhd`**' che si riferisce a un numero intero della dimensione di un '**`signed char`**'; in questi casi occorre comunque considerare che nella trasmissione degli argomenti alle funzioni interviene sempre la promozione a intero, pertanto viene letto il dato della dimensione specificata, ma viene «consumato» il risultato ottenuto dalla promozione.

Simbolo	Tipo	Simbolo	Tipo

`%…hhd`
`%…hhi`
	`signed char`	`%…hhu`
		`%…hho`
		`%…hhx`|`%…hhX`
			`unsigned char`

`%…hd`
`%…hi`
	`short int`	`%…hu`
		`%…ho`
		`%…hx`|`%…hX`
			`unsigned short int`

`%…ld`
`%…li`
	`long int`	`%…lu`
		`%…lo`
		`%…lx`|`%…lX`
			`unsigned long int`

Simbolo	Tipo	Simbolo	Tipo

`%…jd`
`%…ji`
	`intmax_t`	`%…ju`
		`%…jo`
		`%…jx`|`%…jX`
			`uintmax_t`

`%…zd`
`%…zi`
	`size_t`	`%…zu`
		`%…zo`
		`%…zx`|`%…zX`
			`size_t`

`%…td`
`%…ti`
	`ptrdiff_t`	`%…tu`
		`%…to`
		`%…tx`|`%…tX`
			`ptrdiff_t`

Tra il simbolo (*flag*) e il modificatore di lunghezza può apparire un numero che rappresenta l'ampiezza da usare nella trasformazione ed eventualmente la precisione: '***ampiezza*[*.precisione*]**'. Per os16, la precisione si applica esclusivamente alle stringhe, la quale specifica la quantità di caratteri da considerare, troncando il resto.

VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/fprintf.c`' [i161.9.12]

'`lib/stdio/printf.c`' [i161.9.27]

'`lib/stdio/sprintf.c`' [i161.9.34]

'`lib/stdio/snprintf.c`' [i161.9.33]

VEDERE ANCHE

vfprintf(3) [u0.128], *vprintf(3)* [u0.128], *vsprintf(3)* [u0.128], *vsnprintf(3)* [u0.128], *scanf(3)* [u0.90].

os16: process_info(3)

«

NOME

'**`process_info`**' - funzione diagnostica

SINTASSI

```
#include <sys/os16.h>
void process_info (void);
```

DESCRIZIONE

Si tratta di una funzione diagnostica che non richiede argomenti e non restituisce alcunché, per visualizzare, attraverso lo standard output, lo stato dei registri della CPU, i riferimenti principali della collocazione in memoria del processo elaborativo e lo spazio ancora non utilizzato dalla pila dei dati.

Per poter dare un'informazione utile sullo spazio non ancora utilizzato dalla pila dei dati, occorre che prima di questa funzione sia stata chiamata ***heap_clear()***.

FILE SORGENTI

'`lib/sys/os16.h`' [u0.12]

'`lib/sys/os16/process_info.c`' [i161.12.14]

VEDERE ANCHE

cs(3) [u0.12].

ds(3) [u0.12].

es(3) [u0.12].

ss(3) [u0.12].

bp(3) [u0.12].

sp(3) [u0.12].

heap_clear(3) [u0.57].

heap_min(3) [u0.57].

os16: putc(3)

Vedere *fputc(3)* [u0.37].

os16: putchar(3)

Vedere *fputc(3)* [u0.37].

os16: putenv(3)

NOME

'**`putenv`**' - assegnamento di una variabile di ambiente

SINTASSI

```
#include <stdlib.h>
int putenv (const char *string);
```

DESCRIZIONE

La funzione ***putenv()*** assegna una variabile di ambiente. Se questa esiste già, va a rimpiazzare il valore assegnatole in precedenza, altrimenti la crea contestualmente.

La funzione richiede un solo parametro, costituito da una stringa in cui va specificato il nome della variabile e il contenuto da assegnargli, usando la forma '***nome=valore***'. Per esempio, '**`PATH=/bin:/usr/bin`**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l'errore indicato dalla variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/environment.c`' [i161.10.9]

'`lib/stdlib/putenv.c`' [i161.10.14]

VEDERE ANCHE

environ(7) [u0.1], *getenv(3)* [u0.51], *setenv(3)* [u0.94], *unsetenv(3)* [u0.94].

os16: puts(3)

Vedere *fputs(3)* [u0.38].

os16: qsort(3)

NOME

'**`qsort`**' - riordino di un array

SINTASSI

```
#include <stdlib.h>
void qsort (void *base, size_t nmemb, size_t size,
            int (*compare)(const void *, const void *));
```

DESCRIZIONE

La funzione ***qsort()*** riordina un array composto da ***nmemb*** elementi da ***size*** byte ognuno. Il primo argomento, ovvero il parametro ***base***, è il puntatore all'indirizzo iniziale di questo array in memoria.

Il riordino avviene comparando i vari elementi con l'ausilio di una funzione, passata tramite il suo puntatore, la quale deve ricevere due argomenti, costituiti dai puntatori agli elementi dell'array da confrontare. Tale funzione deve restituire un valore minore di zero per un confronto in cui il suo primo argomento deve essere collocato prima del secondo; un valore pari a zero se gli argomenti sono uguali ai fini del riordino; un valore maggiore di zero se il suo primo argomento va collocato dopo il secondo nel riordino.

Segue un esempio di utilizzo della funzione ***qsort()***:

```
#include <stdio.h>
#include <stdlib.h>

int confronta (const void *a, const void *b)
{
    int x = *((int *) a);
    int y = *((int *) b);
    return x - y;
}

int main (void)
{
    int a[] = {3, 1, 5, 2};

    qsort (&a[0], 4, sizeof (int), confronta);
    printf ("%d %d %d %d\n", a[0], a[1], a[2], a[3]);

    return 0;
}
```

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/qsort.c`' [i161.10.15]

os16: rand(3)

NOME

'**`rand`**' - generazione di numeri pseudo-casuali

SINTASSI

```
#include <stdlib.h>
int  rand  (void);
void srand (unsigned int seed);
```

DESCRIZIONE

La funzione ***rand()*** produce un numero intero casuale, sulla base di un seme, il quale può essere cambiato in ogni momento, con

l'ausilio di ***srand()***. A ogni chiamata della funzione ***rand()***, il risultato ottenuto, viene utilizzato anche come seme per la chiamata successiva. Se inizialmente non viene assegnato alcun seme, il primo valore predefinito è pari a 1.

VALORE RESTITUITO

La funzione ***rand()*** restituisce un numero intero casuale, determinato sulla base del seme accumulato in precedenza.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/rand.c`' [i161.10.16]

os16: readdir(3)

«

NOME

'**`readdir`**' - lettura di una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir (DIR *dp);
```

DESCRIZIONE

La funzione ***readdir()*** legge una voce dalla directory rappresentata da ***dp*** e restituisce il puntatore a una variabile strutturata di tipo '**`struct dirent`**', contenente le informazioni tratte dalla voce letta. La variabile strutturata in questione si trova in memoria statica e viene sovrascritta con le chiamate successive della funzione ***readdir()***.

Il tipo '**`struct dirent`**' è definito nel file di intestazione '`dirent.h`', nel modo seguente:

```
struct dirent {
    ino_t       d_ino;
    char        d_name[NAME_MAX+1];
};
```

Il membro ***d_ino*** è il numero di inode del file il cui nome appare nel membro ***d_name***. La macro-variabile ***NAME_MAX*** è dichiarata a sua volta nel file di intestazione '`limits.h`'. La dimensione del membro ***d_name*** è tale da permettere di includere anche il valore zero di terminazione delle stringhe.

VALORE RESTITUITO

La funzione restituisce il puntatore a una variabile strutturata di tipo '**`struct dirent`**'; se la lettura ha già raggiunto la fine della directory, oppure per qualunque altro tipo di errore, la funzione restituisce '**`NULL`**' e aggiorna eventualmente la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	La directory rappresentata da ***dp*** non è valida.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/readdir.c`' [i161.2.4]

VEDERE ANCHE

read(2) [u0.29], *closedir(3)* [u0.10], *opendir(3)* [u0.76], *rewinddir(3)* [u0.89].

os16: realloc(3)

«

Vedere *malloc(3)* [u0.66].

os16: rewind(3)

«

NOME

'**`rewind`**' - riposizionamento all'inizio dell'indice di accesso a un flusso di file

SINTASSI

```
#include <stdio.h>
void rewind (FILE *fp);
```

DESCRIZIONE

La funzione ***rewind()*** azzera l'indice della posizione interna del flusso di file specificato con il parametro ***fp***; inoltre azzera anche l'indicatore di errore dello stesso flusso. In pratica si ottiene la stessa cosa di:

```
(void) fseek (fp, 0L, SEEK_SET);
clearerr (fp);
```

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/rewind.c`' [i161.9.29]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.46], *fseek(3)* [u0.43], *rewind(3)* [u0.88].

os16: rewinddir(3)

«

NOME

'**`rewinddir`**' - riposizionamento all'inizio del riferimento per l'accesso a una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir (DIR *dp);
```

DESCRIZIONE

La funzione ***rewinddir()*** riposiziona i riferimenti per l'accesso alla directory indicata, in modo che la prossima lettura o scrittura avvenga dalla prima posizione.

VALORE RESTITUITO

La funzione non restituisce alcunché e non si presenta nemmeno la possibilità di segnalare errori attraverso la variabile ***errno***.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/rewinddir.c`' [i161.2.5]

VEDERE ANCHE

rewind(3) [u0.88], *closedir(3)* [u0.10], *opendir(3)* [u0.76], *rewinddir(3)* [u0.86].

os16: scanf(3)

«

NOME

'**`scanf`**', '**`fscanf`**', '**`sscanf`**' - interpretazione dell'input e conversione

SINTASSI

```
#include <stdio.h>
int scanf  (const char *restrict format, ...);
int fscanf (FILE *restrict fp,
            const char *restrict format, ...);
int sscanf (char *restrict string,
            const char *restrict format, ...);
```

DESCRIZIONE

Le funzioni del gruppo '…**scanf()**' hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per ***scanf()***, tramite il flusso di file ***fp*** per ***fscanf()***, oppure tramite la stringa ***string*** per ***sscanf()***. L'interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro ***format***, per le tre funzioni. La stringa di formato contiene degli ***specificatori di conversione***, con cui si determina il tipo degli argomenti variabili che non sono esplicitati nel prototipo delle funzioni.

Per ogni specificatore di conversione contenuto nella stringa di formato, deve esistere un argomento, successivo al parametro ***format***, costituito dal puntatore a una variabile di tipo conforme a quanto indicato dallo specificatore relativo. La conversione per quello specificatore, comporta la memorizzazione del risultato in memoria, in corrispondenza del puntatore relativo. Si osservi l'esempio seguente:

```
int valore;
…
scanf ("%i", &valore);
```

In questo modo si attende l'inserimento, attraverso lo standard input, di un numero intero, da convertire e assegnare così alla variabile ***valore***; Infatti, lo specificatore di conversione '**%i**', consente di interpretare un numero intero.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di os16:

```
%[*][n_ampiezza][hh|h|l|j|z|t]tipo
```

Come si può vedere, all'inizio può apparire un asterisco, il cui scopo è quello di annullare l'assegnamento del valore a una variabile. In pratica, con l'asterisco il dato corrispondente allo specificatore viene interpretato, ma poi non viene salvato.

Successivamente può apparire un numero che rappresenta l'ampiezza del dato da interpretare, in byte, il cui scopo è quello di limitare la lettura fino a un certo carattere.

Dopo può apparire una sigla, composta da una o più lettere, il cui scopo è quello di modificare la dimensione predefinita della variabile di destinazione. In altri termini, senza questo modificatore si intende che la variabile ricevente debba essere di una certa grandezza, ma con l'aggiunta del «modificatore di lunghezza» si precisa invece qualcosa di diverso. In pratica, il modificatore di lunghezza usato da queste funzioni è equivalente a quello delle funzioni di composizione dell'output.

Al termine dello specificatore di conversione appare una lettera che dichiara come deve essere interpretato il dato in ingresso e, in mancanza del modificatore di lunghezza, indica anche la dimensione della variabile ricevente.

Tipi di conversione.

Simbolo	Tipo di argomento	Conversione applicata
%…d	int *	Numero intero con segno rappresentato in base dieci.
%…i	int *	Numero intero con segno rappresentare in base dieci o in base otto, avendo come prefisso uno zero, oppure in base sedici, avendo come prefisso '**0x**' o '**0X**'.
%…u	unsigned int *	Numero intero senza segno rappresentato in base dieci.
%…o	unsigned int *	Numero intero senza segno rappresentato in ottale (con o senza lo zero iniziale).

Simbolo	Tipo di argomento	Conversione applicata
%…x	unsigned int *	Numero intero senza segno rappresentato in esadecimale (con o senza il prefisso '**0x**' o '**0X**').
%…c	char *	Interpreta un solo carattere, o più caratteri se si specifica l'ampiezza. Nella lettura contano anche gli spazi o qualunque altro carattere e non viene aggiunto il carattere nullo di terminazione.
%…s	char *	Interpreta una sequenza di caratteri che non siano spazi, aggiungendo alla fine il carattere nullo di terminazione.
%p	void *	Interpreta il valore di un puntatore che sia rappresentato nello stesso modo in cui farebbe la funzione '**printf("%p",** ***puntatore*****)**'.

Simbolo	Tipo di argomento	Conversione applicata
%n	int *	Questo specificatore non esegue alcuna conversione e si limita a memorizzare la quantità di caratteri ('**char**') letti fino a quel punto.
%…[…]	char *	Interpreta una stringa non vuota contenente solo i caratteri elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri si cerca anche la parentesi quadra chiusa, questa va messa all'inizio dell'elenco: '**%…[]…]**'.
%…[^…]	char *	Interpreta una stringa non vuota contenente solo caratteri diversi da quelli elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri da escludere si vuole indicare anche la parentesi quadra chiusa, questa va messa all'inizio dell'elenco: '**%…[^]…]**'.
%%		Interpreta un carattere di percentuale tra i dati in ingresso, ma senza memorizzare alcunché.

Modificatori della lunghezza del dato in uscita.

Simbolo	Tipo	Simbolo	Tipo
%…hhd %…hhi	signed char *	%…hhu %…hho %…hhx %…hhn	unsigned char *
%…hd %…hi	short int *	%…hu %…ho %…hx %…hn	unsigned short int *
%…ld %…li	long int *	%…lu %…lo %…lx %…ln	unsigned long int *

Simbolo	Tipo	Simbolo	Tipo
%…jd %…ji	intmax_t *	%…ju %…jo %…jx %…jn	uintmax_t *
%…zd %…zi	size_t *	%…zu %…zo %…zx %…zn	size_t *
%…td %…ti	ptrdiff_t *	%…tu %…to %…tx %…tn	ptrdiff_t *

La stringa di conversione è composta da ***direttive***, ognuna delle quali è formata da: uno o più spazi (spazi veri e propri o caratteri di tabulazione orizzontale); un carattere diverso da '**%**' e diverso dai caratteri che rappresentano spazi, oppure uno specificatore di conversione.

[*spazi*] *carattere* | %…

Dalla sequenza di caratteri che costituisce i dati in ingresso da interpretare, vengono eliminati automaticamente gli spazi iniziali e finali (tutto ciò che si può considerare spazio, anche il codice di interruzione di riga), quando all'inizio o alla fine non ci sono corrispondenze con specificatori di conversione che possono interpretarli.

Quando la direttiva di interpretazione inizia con uno o più spazi orizzontali, significa che si vogliono ignorare gli spazi a partire dalla posizione corrente nella lettura dei dati in ingresso; inoltre, la presenza di un carattere che non fa parte di uno specificatore di conversione indica che quello stesso carattere deve essere incontrato nell'interpretazione dei dati in ingresso, altrimenti il procedimento di lettura e valutazione si deve interrompere. Se due specificatori di conversione appaiono adiacenti, i dati in ingresso corrispondenti possono essere separati da spazi orizzontali o da spazi verticali (il codice di interruzione di riga).

VALORE RESTITUITO

Le funzioni restituiscono la quantità di elementi in ingresso interpretati e assegnati correttamente: una quantità inferiore al previsto indica pertanto un errore. Se le funzioni restituiscono il valore '**EOF**', si tratta di un errore, dovuto eventualmente a un problema di interpretazione del formato o a un problema di accesso al flusso di file da cui deve provenire l'input.

ERRORI

Valore di *errno*	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione di accesso non consentita.
EACCES	Accesso non consentito.
EBADF	Il descrittore del file a cui si riferisce il flusso, non è valido.
ERANGE	Il risultato della conversione di un intero, non può essere memorizzato nel tipo di variabile a cui si riferisce lo specificatore di conversione.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/fscanf.c' [i161.9.17]

'lib/stdio/scanf.c' [i161.9.30]

'lib/stdio/sscanf.c' [i161.9.35]

VEDERE ANCHE

vfscanf(3) [u0.129], *vscanf(3)* [u0.129], *vsscanf(3)* [u0.129], *printf(3)* [u0.78].

os16: seg_d(3)

NOME

'**seg_d**', '**seg_i**' - collocazione del processo in memoria

SINTASSI

```
#include <sys/os16.h>
unsigned int seg_d (void);
unsigned int seg_i (void);
```

DESCRIZIONE

Le funzioni elencate nel quadro sintattico, sono in realtà delle macroistruzioni, chiamanti funzioni con nomi analoghi, ma preceduti da un trattino basso (***_seg_d()*** e ***_seg_i()***), per interrogare, rispettivamente, lo stato del registro ***DS*** e ***CS***. Questi due registri indicano, rispettivamente, la collocazione dell'area dati e dell'area codice del processo in corso. Eventualmente, per conoscere l'indirizzo efficace di memoria corrispondente, occorre moltiplicare questi valori per 16.

FILE SORGENTI

'lib/sys/os16.h' [u0.12]

'lib/sys/os16/_seg_i.s' [i161.12.6]

'lib/sys/os16/_seg_d.s' [i161.12.5]

VEDERE ANCHE

cs(3) [u0.12].

ds(3) [u0.12].

es(3) [u0.12].

ss(3) [u0.12].

bp(3) [u0.12].

sp(3) [u0.12].

os16: seg_i(3)

Vedere *seg_d(3)* [u0.91].

os16: setbuf(3)

NOME

'**setbuf**', '**setvbuf**' - modifica della memoria tampone per i flussi di file

SINTASSI

```
#include <stdio.h>
void setbuf (FILE *restrict fp, char *restrict buffer);
int setvbuf (FILE *restrict fp, char *restrict buffer,
             int buf_mode, size_t size);
```

DESCRIZIONE

Le funzioni ***setbuf()*** e ***setvbuf()*** della libreria di os16, non fanno alcunché, perché os16 non gestisce una memoria tampone per i flussi di file.

VALORE RESTITUITO

La funzione ***setvbuf()*** restituisce, in tutti i casi, il valore zero.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/setbuf.c' [i161.9.31]

'lib/stdio/setvbuf.c' [i161.9.32]

VEDERE ANCHE

fflush(3) [u0.30].

os16: setenv(3)

NOME

'**setenv**', '**unsetenv**' - assegnamento o cancellazione di una variabile di ambiente

SINTASSI

```
#include <stdlib.h>
int setenv   (const char *name, const char *value,
              int overwrite);
int unsetenv (const char *name);
```

DESCRIZIONE

La funzione ***setenv()*** crea o assegna un valore a una variabile di ambiente. Se questa variabile esiste già, la modifica del valore assegnatole può avvenire soltanto se l'argomento corrispondente al parametro ***overwrite*** risulta essere diverso da zero; in caso contrario, la modifica non ha luogo.

La funzione ***unsetenv()*** si limita a cancellare la variabile di ambiente specificata come argomento.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l'errore indicato dalla variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EINVAL	Argomento non valido.
ENOMEM	Memoria insufficiente.

FILE SORGENTI

'lib/stdlib.h' [u0.10]

'applic/crt0.s' [i162.1.9]

'lib/stdlib/environment.c' [i161.10.9]

'lib/stdlib/setenv.c' [i161.10.17]

'lib/stdlib/unsetenv.c' [i161.10.20]

VEDERE ANCHE

environ(7) [u0.1], *getenv(3)* [u0.51], *putenv(3)* [u0.82].

os16: setpwent(3)

Vedere *getpwent(3)* [u0.53].

os16: setvbuf(3)

Vedere *setbuf(3)* [u0.93].

os16: snprintf(3)

Vedere *printf(3)* [u0.78].

os16: sp(3)

Vedere *cs(3)* [u0.12].

os16: sprintf(3)

Vedere *printf(3)* [u0.78].

os16: srand(3)

Vedere *rand(3)* [u0.85].

os16: ss(3)

Vedere *cs(3)* [u0.12].

os16: sscanf(3)

Vedere *scanf(3)* [u0.90].

os16: stdio(3)

NOME

'**stdio**' - libreria per la gestione dei file in forma di flussi di file (*stream*)

SINTASSI

```
#include <stdio.h>
```

DESCRIZIONE

Le funzioni di libreria che fanno capo al file di intestazione 'stdio.h', consentono di gestire i file in forma di «flussi», rappresentati da puntatori al tipo '**FILE**'. Questa gestione si sovrappone a quella dei file in forma di «descrittori», la quale avviene tramite chiamate di sistema. Lo scopo della sovrapposizione dovrebbe essere quello di gestire i file con l'ausilio di una memoria tampone, cosa che però la libreria di os16 non fornisce.

Nella libreria di os16, il tipo '**FILE ***' è un puntatore a una variabile strutturata che contiene solo tre informazioni: il numero del descrittore del file a cui il flusso si associa; lo stato di errore; lo stato di raggiungimento della fine del file.

```
typedef struct {
    int           fdn;      // File descriptor number.
    char          error;    // Error indicator.
    char          eof;      // End of file indicator.
} FILE;
```

Le variabili strutturate necessarie per questa gestione, sono raccolte in un array, dichiarato nel file 'lib/stdio/FILE.c', con il nome ***_stream[]***, dove per il descrittore di file ***n***, si associano sempre i dati di ***_stream[n]***.

```
FILE _stream[FOPEN_MAX];
```

Così come sono previsti tre descrittori (zero, uno e due) per la gestione di standard input, standard output e standard error, tutti i processi inizializzano l'array ***_stream[]*** con l'abbinamento a tali descrittori, per i primi tre flussi.

```
void
_stdio_stream_setup (void)
{
    _stream[0].fdn   = 0;
    _stream[0].error = 0;
    _stream[0].eof   = 0;

    _stream[1].fdn   = 1;
    _stream[1].error = 0;
    _stream[1].eof   = 0;

    _stream[2].fdn   = 2;
    _stream[2].error = 0;
    _stream[2].eof   = 0;
}
```

Ciò avviene attraverso il codice contenuto nel file 'crt0.s', dove si chiama la funzione che provvede a tale inizializzazione, contenuta nel file 'lib/stdio/FILE.c'. Per fare riferimento ai flussi predefiniti, si usano i nomi '**stdin**', '**stdout**' e '**stderr**', i quali sono dichiarati nel file 'stdio.h', come puntatori ai primi tre elementi dell'array ***_stream[]***:

```
#define stdin   (&_stream[0])
#define stdout  (&_stream[1])
#define stderr  (&_stream[2])
```

FILE SORGENTI

'lib/sys/types.h' [u0.14]

‘lib/stdio.h’ [u0.9]

‘lib/stdio/FILE.c’ [i161.9.1]

‘applic/crt0.s’ [i162.1.9]

VEDERE ANCHE

close(2) [u0.7], *open(2)* [u0.28], *read(2)* [u0.29], *write(2)* [u0.44].

os16: strcat(3)

NOME

‘**strcat**’, ‘**strncat**’ - concatenamento di una stringa a un’altra già esistente

SINTASSI

```
#include <string.h>
char *strcat  (char *restrict dst,
               const char *restrict org);
char *strncat (char *restrict dst,
               const char *restrict org,
               size_t n);
```

DESCRIZIONE

Le funzioni ***strcat()*** e ***strncat()*** copiano la stringa di origine ***org***, aggiungendola alla stringa di destinazione ***dst***, nel senso che la scrittura avviene a partire dal codice di terminazione ‘**\0**’ che viene così sovrascritto. Al termine della copia, viene aggiunto nuovamente il codice di terminazione di stringa ‘**\0**’, nella nuova posizione conclusiva.

Nel caso particolare di ***strncat()***, la copia si arresta al massimo dopo il trasferimento di ***n*** caratteri. Pertanto, la stringa di origine per ***strncat()*** potrebbe anche non essere terminata correttamente, se raggiunge o supera la dimensione di ***n*** caratteri. In ogni caso, nella destinazione viene aggiunto il codice nullo di terminazione di stringa, dopo la copia del carattere ***n***-esimo.

VALORE RESTITUITO

Le due funzioni restituiscono ***dst***.

FILE SORGENTI

‘lib/string.h’ [u0.11]

‘lib/string/strcat.c’ [i161.11.7]

‘lib/string/strncat.c’ [i161.11.16]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: strchr(3)

NOME

‘**strchr**’, ‘**strrchr**’ - ricerca di un carattere all’interno di una stringa

SINTASSI

```
#include <string.h>
char *strchr  (const char *string, int c);
char *strrchr (const char *string, int c);
```

DESCRIZIONE

Le funzioni ***strchr()*** e ***strrchr()*** scandiscono la stringa ***string*** alla ricerca di un carattere uguale al valore di ***c***. La funzione ***strchr()*** scandisce a partire da «sinistra», ovvero ricerca la prima corrispondenza con il carattere ***c***, mentre la funzione ***strrchr()*** cerca l’ultima corrispondenza con il carattere ***c***, pertanto è come se scandisse da «destra».

VALORE RESTITUITO

Se le due funzioni trovano il carattere che cercano, ne restituiscono il puntatore, altrimenti restituiscono ‘**NULL**’.

FILE SORGENTI

‘lib/string.h’ [u0.11]

‘lib/string/strchr.c’ [i161.11.8]

‘lib/string/strrchr.c’ [i161.11.20]

VEDERE ANCHE

memchr(3) [u0.68], *strlen(3)* [u0.112], *strpbrk(3)* [u0.116], *strspn(3)* [u0.118].

os16: strcmp(3)

NOME

‘**strcmp**’, ‘**strncmp**’ - confronto di due stringhe

SINTASSI

```
#include <string.h>
int strcmp  (const char *string1, const char *string2);
int strncmp (const char *string1, const char *string2,
             size_t n);
int strcoll (const char *string1, const char *string2);
```

DESCRIZIONE

Le funzioni ***strcmp()*** e ***strncmp()*** confrontano due stringhe, nel secondo caso, il confronto avviene al massimo fino al ***n***-esimo carattere.

La funzione ***strcoll()*** dovrebbe eseguire il confronto delle due stringhe tenendo in considerazione la configurazione locale. Tuttavia, os16 non è in grado di gestire le configurazioni locali, pertanto questa funzione coincide esattamente con ***strcmp()***.

VALORE RESTITUITO

Valore	Esito del confronto.
-1	***string1*** < ***string2***
0	***string1*** == ***string2***
+1	***string1*** > ***string2***

FILE SORGENTI

‘lib/string.h’ [u0.11]

‘lib/string/strcmp.c’ [i161.11.9]

‘lib/string/strncmp.c’ [i161.11.17]

‘lib/string/strcoll.c’ [i161.11.10]

VEDERE ANCHE

memcmp(3) [u0.69].

os16: strcoll(3)

Vedere *strcmp(3)* [u0.106].

os16: strcpy(3)

NOME

‘**strcpy**’, ‘**strncpy**’ - copia di una stringa

SINTASSI

```
#include <string.h>
char *strcpy  (char *restrict dst,
               const char *restrict org);
char *strncpy (char *restrict dst,
               const char *restrict org,
               size_t n);
```

DESCRIZIONE

Le funzioni ***strcpy()*** e ***strncpy()***, copiano la stringa ***org***, completa di codice nullo di terminazione, nella destinazione ***dst***. Eventualmente, nel caso di ***strncpy()***, la copia non supera i primi ***n*** caratteri, con l’aggravante che in tal caso, se nei primi ***n*** caratteri non c’è il codice nullo di terminazione delle stringhe, nella destinazione ***dst*** si ottiene una stringa non terminata.

VALORE RESTITUITO

Le funzioni restituiscono ***dst***.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/strcpy.c`’ [i161.11.11]

‘`lib/string/strncpy.c`’ [i161.11.18]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *memmove(3)* [u0.71].

os16: strcspn(3)

Vedere *strspn(3)* [u0.118].

os16: strdup(3)

NOME

‘**`strdup`**’ - duplicazione di una stringa

SINTASSI

```
#include <string.h>
char *strdup  (const char *string);
```

DESCRIZIONE

La funzione ***strdup()***, alloca dinamicamente una quantità di memoria, necessaria a copiare la stringa ***string***, quindi esegue tale copia e restituisce il puntatore alla nuova stringa allocata. Tale puntatore può essere usato successivamente per liberare la memoria, con l’ausilio della funzione ***free()***.

VALORE RESTITUITO

La funzione restituisce il puntatore alla nuova stringa ottenuta dalla copia, oppure ‘**`NULL`**’ nel caso non fosse possibile allocare la memoria necessaria.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/strdup.c`’ [i161.11.13]

VEDERE ANCHE

free(3) [u0.66], *malloc(3)* [u0.66], *realloc(3)* [u0.66].

os16: strerror(3)

NOME

‘**`strerror`**’ - descrizione di un errore in forma di stringa

SINTASSI

```
#include <string.h>
char *strerror (int errnum);
```

DESCRIZIONE

La funzione ***strerror()*** interpreta il valore ***errnum*** come un errore, di quelli che può rappresentare la variabile ***errno*** del file ‘`errno.h`’.

VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa contenente la descrizione dell’errore, oppure soltanto ‘**`Unknown error`**’, se l’argomento ricevuto non è traducibile.

FILE SORGENTI

‘`lib/errno.h`’ [u0.3]

‘`lib/string.h`’ [u0.11]

‘`lib/string/strerror.c`’ [i161.11.14]

VEDERE ANCHE

errno(3) [u0.18], *perror(3)* [u0.77].

os16: strlen(3)

NOME

‘**`strlen`**’ - lunghezza di una stringa

SINTASSI

```
#include <string.h>
size_t strlen (const char *string);
```

DESCRIZIONE

La funzione ***strlen()*** calcola la lunghezza della stringa, ovvero la quantità di caratteri che la compone, escludendo il codice nullo di conclusione.

VALORE RESTITUITO

La funzione restituisce la quantità di caratteri che compone la stringa, escludendo il codice ‘**`\0`**’ finale.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/strlen.c`’ [i161.11.15]

os16: strncat(3)

Vedere *strcat(3)* [u0.104].

os16: strncmp(3)

Vedere *strcmp(3)* [u0.106].

os16: strncpy(3)

Vedere *strcpy(3)* [u0.108].

os16: strpbrk(3)

NOME

‘**`strpbrk`**’ - scansione di una stringa alla ricerca di un carattere

SINTASSI

```
#include <string.h>
char *strpbrk (const char *string, const char *accept);
```

DESCRIZIONE

La funzione ***strpbrk()*** cerca il primo carattere, nella stringa ***string***, che corrisponda a uno di quelli contenuti nella stringa ***accept***.

VALORE RESTITUITO

Restituisce il puntatore al primo carattere che, nella stringa ***string*** corrisponde a uno di quelli contenuti nella stringa ***accept***. In mancanza di alcuna corrispondenza, restituisce ‘**`NULL`**’.

FILE SORGENTI

‘`lib/string.h`’ [u0.11]

‘`lib/string/strpbrk.c`’ [i161.11.19]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strstr(3)* [u0.119], *strtok(3)* [u0.120].

os16: strrchr(3)

Vedere *strchr(3)* [u0.105].

os16: strspn(3)

NOME

'**strspn**', '**strcspn**' - scansione di una stringa, limitatamente a un certo insieme di caratteri

SINTASSI

```
#include <string.h>
size_t strspn  (const char *string, const char *accept);
size_t strcspn (const char *string, const char *reject);
```

DESCRIZIONE

La funzione ***strspn()*** scandisce la stringa ***string***, calcolando la lunghezza di questa che contiene, a partire dall'inizio, soltanto caratteri che si trovano nella stringa ***accept***.

La funzione ***strcspn()*** scandisce la stringa ***string***, calcolando la lunghezza di questa che contiene, a partire dall'inizio, soltanto caratteri che non si trovano nella stringa ***reject***.

VALORE RESTITUITO

La funzione ***strspn()*** restituisce la lunghezza della stringa che contiene soltanto caratteri contenuti in ***accept***.

La funzione ***strcspn()*** restituisce la lunghezza della stringa che contiene soltanto caratteri che non sono contenuti in ***reject***.

FILE SORGENTI

'`lib/string.h`' [u0.11]

'`lib/string/strspn.c`' [i161.11.21]

'`lib/string/strcspn.c`' [i161.11.12]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strstr(3)* [u0.119], *strtok(3)* [u0.120].

os16: strstr(3)

NOME

'**strstr**' - ricerca di una sottostringa

SINTASSI

```
#include <string.h>
char *strstr (const char *string, const char *substring);
```

DESCRIZIONE

La funzione ***strstr()*** scandisce la stringa ***string***, alla ricerca della prima corrispondenza con la stringa ***substring***, restituendo eventualmente il puntatore all'inizio di tale corrispondenza.

VALORE RESTITUITO

Se la ricerca termina con successo, viene restituito il puntatore all'inizio della sottostringa contenuta in ***string***; diversamente viene restituito il puntatore nullo '**NULL**'.

FILE SORGENTI

'`lib/string.h`' [u0.11]

'`lib/string/strstr.c`' [i161.11.22]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strtok(3)* [u0.120].

os16: strtok(3)

NOME

'**strtok**' - *string token*, ovvero estrazione di pezzi da una stringa

SINTASSI

```
#include <string.h>
char *strtok (char *restrict string,
              const char *restrict delim);
```


DESCRIZIONE

La funzione ***strtok()*** serve a suddividere una stringa in unità, definite *token*, specificando un elenco di caratteri da intendere come delimitatori, in una seconda stringa. La funzione va usata in fasi successive, fornendo solo inizialmente la stringa da suddividere che continua poi a essere utilizzata se al suo posto viene fornito il puntatore nullo. La funzione restituisce, di volta in volta, il puntatore alla sottostringa contenente l'unità individuata, oppure il puntatore nullo, se non può trovarla.

La funzione deve tenere memoria di un puntatore in modo persistente e deve isolare le unità modificando la stringa originale, inserendo il carattere nullo di terminazione alla fine delle unità individuate.

Quando la funzione viene chiamata indicando al posto della stringa da scandire il puntatore nullo, l'insieme dei delimitatori può essere diverso da quello usato nelle fasi precedenti.

Per comprendere lo scopo della funzione viene utilizzato lo stesso esempio che appare nel documento *ISO/IEC 9899:TC2*, al paragrafo 7.21.5.7, con qualche piccola modifica per poterlo rendere un programma autonomo:

```
#include <stdio.h>
#include <string.h>
int
main (void)
{
  char str[] = "?a???b,,,#c";
  char *t;

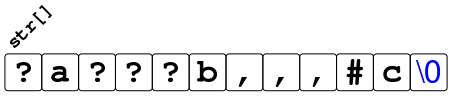
  t = strtok (str, "?");          // t punta all'unità "a"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, ",");         // t punta all'unità "??b"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, "#,");        // t punta all'unità "c"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, "?");         // t è un puntatore nullo
  printf ("strtok: \"%s\"\n", t);

  return 0;
}
```

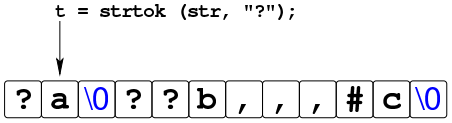
Avviando il programma si ottiene quanto già descritto dai commenti inseriti nel codice:

```
strtok: "a"
strtok: "??b"
strtok: "c"
strtok: "(null)"
```

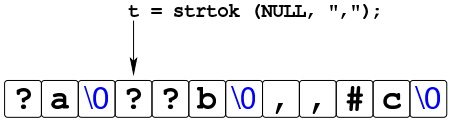
Ciò che avviene nell'esempio può essere schematizzato come segue. Inizialmente la stringa '**str**' ha in memoria l'aspetto seguente:



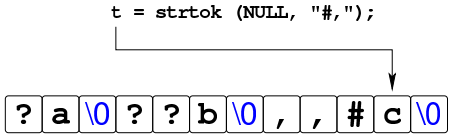
Dopo la prima chiamata della funzione ***strtok()*** la stringa risulta alterata e il puntatore ottenuto raggiunge la lettera '**a**':



Dopo la seconda chiamata della funzione, in cui si usa il puntatore nullo per richiedere una scansione ulteriore della stringa originale, si ottiene un nuovo puntatore che, questa volta, inizia a partire dal quarto carattere, rispetto alla stringa originale, dal momento che il terzo è già stato sovrascritto da un carattere nullo:



La penultima chiamata della funzione ***strtok()*** raggiunge la lettera '**c**' che è anche alla fine della stringa originale:



L'ultimo tentativo di chiamata della funzione non può dare alcun esito, perché la stringa originale si è già conclusa.

VALORE RESTITUITO

La funzione restituisce il puntatore al prossimo «pezzo«, oppure '**NULL**' se non ce ne sono più.

FILE SORGENTI

'`lib/string.h`' [u0.11]

'`lib/string/strtok.c`' [i161.11.23]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strspn(3)* [u0.118].

os16: strtol(3)

NOME

'**strtol**', '**strtoul**' - conversione di una stringa in un numero

SINTASSI

```
#include <stdlib.h>
long int strtol (const char *restrict string,
                 char **restrict endptr,
                 int base);

unsigned long int strtoul (const char *restrict string,
                           char **restrict endptr,
                           int base);
```

DESCRIZIONE

Le funzioni ***strtol()*** e ***strtoul()***, convertono la stringa ***string*** in un numero, intendendo la sequenza di caratteri nella base di numerazione indicata come ultimo argomento (***base***). Tuttavia, la base di numerazione potrebbe essere omessa (valore zero) e in tal caso la stringa deve essere interpretata ugualmente in qualche modo: se (dopo un segno eventuale) inizia con zero seguito da un'altra cifra numerica, deve trattarsi di una sequenza ottale; se inizia con zero, quindi appare una lettera «x» deve trattarsi di un numero esadecimale; se inizia con una cifra numerica diversa da zero, deve trattarsi di un numero in base dieci.

La traduzione della stringa ha luogo progressivamente, arrestandosi quando si incontra un carattere incompatibile con la base di numerazione selezionata o stabilita automaticamente. Il valore convertito viene restituito; inoltre, se il puntatore ***endptr*** è valido (diverso da '**NULL**'), si assegna a ****endptr*** la posizione raggiunta nella stringa, corrispondente al primo carattere che non può essere convertito. Pertanto, nello stesso modo, se la stringa non può essere convertita affatto e si può assegnare qualcosa a ****endptr***, alla fine, ****endptr*** corrisponde esattamente a ***string***.

VALORE RESTITUITO

Le funzioni restituiscono il valore tratto dall'interpretazione della stringa, ammesso che sia rappresentabile, altrimenti si ottiene '**LONG_MIN**' o '**LONG_MAX**', a seconda dei casi, sapendo che occorre consultare la variabile ***errno*** per maggiori dettagli.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ERANGE`	Il valore risultante è al di fuori dell'intervallo ammissibile per la rappresentazione.

DIFETTI

La realizzazione di ***strtoul()*** è incompleta, in quanto si limita a utilizzare ***strtol()***, convertendo il risultato in un valore senza segno.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/strtol.c`' [i161.10.18]

'`lib/stdlib/strtoul.c`' [i161.10.19]

os16: strtoul(3)

Vedere *strtol(3)* [u0.121].

os16: strxfrm(3)

NOME

'**strxfrm**' - *string transform*, ovvero trasformazione di una stringa

SINTASSI

```
#include <string.h>
size_t strxfrm (char *restrict dst,
                const char *restrict org,
                size_t n);
```

DESCRIZIONE

Lo scopo della funzione ***strxfrm()*** sarebbe quello di copiare la stringa ***org***, sovrascrivendo ***dst***, fino a un massimo di ***n*** caratteri nella destinazione, ma applicando una trasformazione relativa alla configurazione locale.

os16 non gestisce la configurazione locale, pertanto questa funzione si comporta in modo simile a ***strncpy()***, con una differenza in ciò che viene restituito.

VALORE RESTITUITO

La funzione restituisce la quantità di byte utilizzati per contenere la trasformazione in ***dst***, senza però contare il carattere nullo di terminazione.

FILE SORGENTI

'`lib/string.h`' [u0.11]

'`lib/string/strxfrm.c`' [i161.11.24]

VEDERE ANCHE

memcmp(3) [u0.69], *strcmp(3)* [u0.106], *strcoll(3)* [u0.106].

os16: ttyname(3)

NOME

'**ttyname**' - determinazione del percorso del file di dispositivo di un terminale aperto

SINTASSI

```
#include <unistd.h>
char *ttyname (int fdn);
```

DESCRIZIONE

La funzione ***ttyname()*** richiede come unico argomento il numero che identifica il descrittore di un file. Ammesso che tale descrittore si riferisca a un terminale, la funzione restituisce il puntatore a una stringa che rappresenta il percorso del file di dispositivo corrispondente.

La stringa in questione viene modificata se si usa la funzione in altre occasioni.

VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa che descrive il percorso del file di dispositivo, presunto, del terminale aperto con il numero ***fdn***. Se non si tratta di un terminale, si ottiene un errore. In ogni caso, se la funzione non può restituire un'informazione corretta, produce semplicemente il puntatore nullo e aggiorna la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file indicato non è valido.
`ENOTTY`	Il descrittore di file indicato non riguarda un terminale.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/ttyname.c`' [i161.17.34]

VEDERE ANCHE

stat(2) [u0.36], *isatty(3)* [u0.61].

os16: unsetenv(3)

«

Vedere *setenv(3)* [u0.94].

os16: vfprintf(3)

«

Vedere *vprintf(3)* [u0.128].

os16: vfscanf(3)

«

Vedere *vfscanf(3)* [u0.129].

os16: vprintf(3)

«

NOME

'**`vprintf`**', '**`vfprintf`**', '**`vsprintf`**', '**`vsnprintf`**' - composizione dei dati per la visualizzazione

SINTASSI

```
#include <stdarg.h>
#include <stdio.h>
int vprintf   (char *restrict format, va_list arg);
int vfprintf  (FILE *fp, char *restrict format,
               va_list arg);
int vsnprintf (char *restrict string, size_t size,
               const char *restrict format, va_list ap);
int vsprintf  (char *string, char *restrict format,
               va_list arg);
```

DESCRIZIONE

Le funzioni del gruppo '**`v`**...**`printf()`**' hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e vengono comunicati attraverso un puntatore di tipo '**`va_list`**'. Per quantificare e qualificare questi dati in ingresso, la stringa a cui punta il parametro ***format***, deve contenere degli ***specificatori di conversione***, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta ***format***, la interpretano e determinano come scandire gli argomenti a cui fa riferimento il puntatore ***arg***, quindi producono un'altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi.

In generale, le funzioni '**`v`**...**`printf()`**' servono per realizzare le altre funzioni '...**`printf()`**', le quali invece ricevono gli argomenti variabili direttamente. Per esempio, la funzione ***printf()*** può essere realizzata utilizzando in pratica ***vprintf()***:


```
#include <stdio.h>
#include <stdarg.h>
int
printf (char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return (vprintf (format, ap));
}
```

Si veda *printf(3)* [u0.78], per la descrizione di come va predisposta la stringa ***format***. Nella realizzazione di os16, di tutte queste funzioni, quella che compie effettivamente il lavoro di interpretazione della stringa di formato e che in qualche modo viene chiamata da tutte le altre, è soltanto ***vsnprintf()***.

VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

FILE SORGENTI

'`lib/stdarg.h`' [i161.1.12]

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/vfprintf.c`' [i161.9.36]

'`lib/stdio/vprintf.c`' [i161.9.39]

'`lib/stdio/vsprintf.c`' [i161.9.42]

'`lib/stdio/vsnprintf.c`' [i161.9.41]

VEDERE ANCHE

fprintf(3) [u0.78], *printf(3)* [u0.78], *sprintf(3)* [u0.78], *snprintf(3)* [u0.78], *scanf(3)* [u0.90].

os16: vscanf(3)

«

NOME

'**`vscanf`**', '**`vfscanf`**', '**`vsscanf`**' - interpretazione dell'input e conversione

SINTASSI

```
#include <stdarg.h>
#include <stdio.h>
int vscanf  (const char *restrict format, va_list ap);
int vfscanf (FILE *restrict fp, const char *restrict format,
             va_list ap);
int vsscanf (const char *string, const char *restrict format,
             va_list ap);
```

DESCRIZIONE

Le funzioni del gruppo '**`v`**...**`scanf()`**' hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per ***vscanf()***, tramite il flusso di file ***fp*** per ***vfscanf()***, oppure tramite la stringa ***string*** per ***vsscanf()***. L'interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro ***format***, per le tre funzioni. La stringa di formato contiene degli ***specificatori di conversione***, con cui si determina il tipo degli argomenti variabili a cui punta inizialmente ***ap***.

Queste funzioni servono per realizzare in pratica quelle corrispondenti che hanno nomi privi della lettera «v» iniziale. Per esempio, per ottenere ***scanf()*** si può utilizzare ***vscanf()***:

```
#include <stdio.h>
#include <stdarg.h>
int
scanf (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vscanf (format, ap);
}
```

Il modo in cui va predisposta la stringa di formato (***format***) è descritto in *scanf(3)* [u0.90]. La funzione più importante di questo gruppo, in quanto svolge effettivamente il lavoro di interpretazione e viene chiamata, più o meno indirettamente, da tutte le altre, è ***vfsscanf()***, la quale però non è standard.

VALORE RESTITUITO

Le funzioni restituiscono la quantità di elementi in ingresso interpretati e assegnati correttamente: una quantità inferiore al previsto indica pertanto un errore. Se le funzioni restituiscono il valore '`EOF`', si tratta di un errore, dovuto eventualmente a un problema di interpretazione del formato o a un problema di accesso al flusso di file da cui deve provenire l'input.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione di accesso non consentita.
`EACCES`	Accesso non consentito.
`EBADF`	Il descrittore del file a cui si riferisce il flusso, non è valido.
`ERANGE`	Il risultato della conversione di un intero, non può essere memorizzato nel tipo di variabile a cui si riferisce lo specificatore di conversione.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/vfscanf.c`' [i161.9.37]

'`lib/stdio/vscanf.c`' [i161.9.40]

'`lib/stdio/vsscanf.c`' [i161.9.43]

'`lib/stdio/vfsscanf.c`' [i161.9.38]

VEDERE ANCHE

fscanf(3) [u0.90], *scanf(3)* [u0.90], *sscanf(3)* [u0.90], *printf(3)* [u0.78].

os16: vsnprintf(3)

Vedere *vprintf(3)* [u0.128].

os16: vsprintf(3)

Vedere *vprintf(3)* [u0.128].

os16: vsscanf(3)

Vedere *vsscanf(3)* [u0.129].

Sezione 4: file speciali

os16: console(4) 1493
os16: dsk(4) 1493
os16: kmem_file(4) 1494
os16: kmem_inode(4) 1494
os16: kmem_mmp(4) 1494
os16: kmem_ps(4) 1495
os16: kmem_sb(4) 1495
os16: mem(4) 1496
os16: null(4) 1496
os16: port(4) 1496
os16: tty(4) 1496
os16: zero(4) 1497

`/dev/console` 1493 `/dev/dsk0` 1493 `/dev/dsk1` 1493
`/dev/kmem_file` 1494 `/dev/kmem_inode` 1494
`/dev/kmem_mmp` 1494 `/dev/kmem_ps` 1495
`/dev/kmem_sb` 1495 `/dev/mem` 1496 `/dev/null` 1496
`/dev/port` 1496 `/dev/tty` 1496 `/dev/zero` 1497

os16: console(4)

NOME

'`/dev/console`' - file di dispositivo che rappresenta la console e le console virtuali

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/console`'	file di dispositivo a caratteri	5	255	0644_8
'`/dev/console0`'	file di dispositivo a caratteri	5	0	0644_8
'`/dev/console1`'	file di dispositivo a caratteri	5	1	0644_8
'`/dev/console2`'	file di dispositivo a caratteri	5	2	0644_8
'`/dev/console3`'	file di dispositivo a caratteri	5	3	0644_8

DESCRIZIONE

Il file di dispositivo '`/dev/console`' rappresenta la console virtuale attiva in un certo momento; i file '`/etc/console`***n***' rappresentano la console virtuale ***n***, dove ***n*** va da zero a quattro.

I permessi di accesso a questi file di dispositivo sono limitati in modo da consentire solo al proprietario di accedere in scrittura. Tuttavia, per i file di dispositivo usati effettivamente come terminali di controllo, i permessi e la proprietà sono gestiti automaticamente dai programmi '**`getty`**' e '**`login`**'.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *tty(4)* [u0.11].

os16: dsk(4)

NOME

'`/dev/dsk`***n***' - file di dispositivo per le unità di memorizzazione a disco

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/dsk0’	file di dispositivo a blocchi	3	0	0644_8
‘/dev/dsk1’	file di dispositivo a blocchi	3	1	0644_8
‘/dev/dsk2’	file di dispositivo a blocchi	3	2	0644_8
‘/dev/dsk3’	file di dispositivo a blocchi	3	3	0644_8

DESCRIZIONE

I file di dispositivo ‘/dev/dsk***n***’ rappresentano, ognuno, un’unità di memorizzazione a disco. La prima unità è ‘/dev/dsk0’, quelle successive procedono con la numerazione.

os16 gestisce solo unità a dischetti da 1440 Kibyte; inoltre, non è ammissibile la suddivisione in partizioni e, in pratica, sono gestibili solo due unità. Pertanto, sono utili solo ‘/dev/dsk0’ e ‘/dev/dsk1’.

VEDERE ANCHE

MAKEDEV(8) [u0.3].

os16: kmem_file(4)

NOME

‘/dev/kmem_file’ - accesso alla memoria del kernel contenente la tabella dei file

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_file’	file di dispositivo a caratteri	4	5	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_file’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella dei file. La tabella dei file è un array di ‘**FILE_MAX_SLOTS**’ elementi, di tipo ‘**file_t**’, secondo le definizioni contenute nel file ‘kernel/fs.h’.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *kmem_ps(4)* [u0.6], *kmem_mmp(4)* [u0.5], *kmem_sb(4)* [u0.7], *kmem_inode(4)* [u0.4].

os16: kmem_inode(4)

NOME

‘/dev/kmem_inode’ - accesso alla memoria del kernel contenente la tabella degli inode

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_inode’	file di dispositivo a caratteri	4	4	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_inode’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella degli inode. La tabella degli inode è un array di ‘**INODE_MAX_SLOTS**’ elementi, di tipo ‘**inode_t**’, secondo le definizioni contenute nel file ‘kernel/fs.h’.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *kmem_ps(4)* [u0.6], *kmem_mmp(4)* [u0.5], *kmem_sb(4)* [u0.7], *kmem_file(4)* [u0.3].

os16: kmem_mmp(4)

NOME

‘/dev/kmem_mmp’ - accesso alla memoria del kernel contenente la mappa di utilizzo della memoria

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_mmp’	file di dispositivo a caratteri	4	2	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_mmp’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la mappa di utilizzo della memoria.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *kmem_ps(4)* [u0.6], *kmem_sb(4)* [u0.7], *kmem_inode(4)* [u0.4], *kmem_file(4)* [u0.3].

os16: kmem_ps(4)

NOME

‘/dev/kmem_ps’ - accesso alla memoria del kernel contenente lo stato dei processi

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_ps’	file di dispositivo a caratteri	4	1	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_ps’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella dei processi. La tabella dei processi è un array di ‘**PROCESS_MAX**’ elementi, di tipo ‘**proc_t**’, secondo le definizioni contenute nel file ‘kernel/proc.h’. Questo meccanismo viene usato dal programma ‘**ps**’ per leggere e visualizzare lo stato dei processi.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *kmem_mmp(4)* [u0.5], *kmem_sb(4)* [u0.7], *kmem_inode(4)* [u0.4], *kmem_file(4)* [u0.3].

os16: kmem_sb(4)

NOME

‘/dev/kmem_sb’ - accesso alla memoria del kernel contenente la tabella dei super blocchi

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_sb’	file di dispositivo a caratteri	4	3	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_sb’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella dei super blocchi. La tabella dei super blocchi è un array di ‘**SB_MAX_SLOTS**’ elementi, di tipo ‘**sb_t**’, secondo le definizioni contenute nel file ‘kernel/fs.h’.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *kmem_ps(4)* [u0.6], *kmem_mmp(4)* [u0.5], *kmem_inode(4)* [u0.4], *kmem_file(4)* [u0.3].

os16: mem(4)

NOME

'`/dev/mem`' - file di dispositivo per l'acceso alla memoria del processo

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/mem`'	file di dispositivo a caratteri	1	1	0444_8

DESCRIZIONE

Il file di dispositivo '`/dev/mem`' consente di leggere la memoria del processo.

VEDERE ANCHE

MAKEDEV(8) [u0.3].

os16: null(4)

NOME

'`/dev/null`' - file di dispositivo per la distruzione dei dati

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/null`'	file di dispositivo a caratteri	1	2	0666_8

DESCRIZIONE

Il file di dispositivo '`/dev/null`' appare in lettura come un file completamente vuoto, mentre in scrittura è un file in cui si può scrivere indefinitivamente, perdendo però i dati che vi si immettono.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *zero(4)* [u0.12].

os16: port(4)

NOME

'`/dev/port`' - file di dispositivo per accedere alle porte di I/O

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/port`'	file di dispositivo a caratteri	1	3	0644_8

DESCRIZIONE

Il file di dispositivo '`/dev/port`' consente di accedere alle porte di I/O. Tali porte consentono di leggere uno o al massimo due byte, pertanto la dimensione della lettura può essere '**`(size_t) 1`**' oppure '**`(size_t) 2`**'. Per selezionare l'indirizzo della porta occorre posizionare il riferimento interno al file a un indirizzo pari a quello della porta, prima di eseguire la lettura o la scrittura.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *mem(4)* [u0.8].

os16: tty(4)

NOME

'`/dev/tty`' - file di dispositivo che rappresenta il terminale di controllo del processo

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/tty`'	file di dispositivo a caratteri	2	0	0666_8

DESCRIZIONE

Il file di dispositivo '`/dev/tty`' rappresenta il terminale di controllo del processo; in altri termini, il processo che accede al file '`/dev/tty`', raggiunge il proprio terminale di controllo.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *console(4)* [u0.1].

os16: zero(4)

NOME

'`/dev/zero`' - file di dispositivo per la produzione del valore zero

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/zero`'	file di dispositivo a caratteri	1	4	0666_8

DESCRIZIONE

Il file di dispositivo '`/dev/zero`' appare in lettura come un file di lunghezza indefinita, contenente esclusivamente il valore zero (lo zero binario), mentre in scrittura è un file in cui si può scrivere indefinitivamente, perdendo però i dati che vi si immettono.

VEDERE ANCHE

MAKEDEV(8) [u0.3], *null(4)* [u0.9].

os16: inittab(5) 1499
os16: issue(5) 1499
os16: passwd(5) 1499

`/etc/inittab` 1499 `/etc/issue` 1499 `/etc/passwd` 1499

os16: inittab(5)

NOME

'`/etc/inittab`' - configurazione di '**`init`**'

DESCRIZIONE

Il file '`/etc/inittab`' contiene la configurazione di '**`init`**', per la definizione dei processi da avviare per la messa in funzione del sistema operativo. Il file può contenere dei commenti, preceduti dal carattere «#» e «voci» costituite da righe suddivise in quattro campi, separati da due punti (:), come nell'esempio seguente:

```
c0:1:respawn:/bin/getty /dev/console0
c1:1:respawn:/bin/getty /dev/console1
```

I campi hanno il significato descritto nell'elenco seguente:

1. codice che identifica univocamente la voce;
2. i livelli di esecuzione per cui la voce è valida;
3. l'azione da compiere sulla voce;
4. il programma da avviare, con tutte le opzioni e gli argomenti necessari.

Il programma '**`init`**' di os16 non distingue i livelli di esecuzione e considera soltanto l'azione '**`respawn`**', con la quale si intende che '**`init`**' debba riavviare il processo, quando questo muore, o comunque quando muore quel processo che ha preso il suo posto.

VEDERE ANCHE

init(8) [u0.2], *getty(8)* [u0.1], *login(1)* [u0.12].

os16: issue(5)

NOME

'`/etc/issue`' - messaggio che precede '**`login`**'

DESCRIZIONE

Il file '`/etc/issue`' viene visualizzato da '**`getty`**', prima dell'avvio di '**`login`**'. Il contenuto predefinito di questo file, per os16, è il seguente:

```
os16: a basic os. [Ctrl q], [Ctrl r], [Ctrl s], [Ctrl t] to change console.
```

Il programma '**`getty`**' di os16 non è in grado di interpretare il contenuto del file, pertanto lo visualizza letteralmente; tuttavia, '**`getty`**' mostra, indipendentemente dalla presenza e dal contenuto del file '`/etc/issue`', delle informazioni sul terminale per il quale è in funzione.

VEDERE ANCHE

getty(8) [u0.1].

os16: passwd(5)

NOME

'`/etc/passwd`' - elenco delle utenze

DESCRIZIONE

Il file '`/etc/passwd`' contiene l'elenco degli utenti del sistema, uno per ogni riga. Le righe sono divise in sette campi, delimitati con il carattere due punti (:), come nell'esempio seguente, che rappresenta l'impostazione predefinita di os16:

```
root:ciao:0:0:root:/root:/bin/shell
user:ciao:1001:1001:test user:/home/user:/bin/shell
```

I campi hanno il significato descritto nell'elenco seguente:

1. nominativo utente;
2. parola d'ordine, in chiaro, per l'identificazione con il programma '**`login`**';
3. numero UID, ovvero il numero dell'utente;
4. numero GID, ovvero il numero del gruppo, ma non utilizzato da os16;
5. descrizione dell'utenza;
6. shell.

Trattandosi di un sistema operativo elementare, la parola d'ordine appare in chiaro nel secondo campo, senza altri accorgimenti. Inoltre, il file deve essere accessibile in lettura a tutti gli utenti.

VEDERE ANCHE

login(1) [u0.12].

Sezione 7: varie

os16: environ(7) 1501
os16: undocumented(7) 1501

`environ` 1501

os16: environ(7)

NOME

'**`environ`**' - ambiente del processo elaborativo

SINTASSI

```
extern char **environ;
```

DESCRIZIONE

La variabile ***environ***, dichiarata nel file '`unistd.h`', punta a un array di stringhe, ognuna delle quali rappresenta una variabile di ambiente, con il valore a lei assegnato. Pertanto, il contenuto di queste stringhe ha una forma del tipo '***nome=valore***'. Per esempio '**`HOME=/home/user`**'.

In generale, l'accesso diretto ai contenuti di questo array non è conveniente, in quanto sono disponibili delle funzioni che facilitano la gestione di questi dati in forma di variabili di ambiente.

Dal momento che le funzioni di accesso alle informazioni sulle variabili di ambiente sono definite nel file '`stdlib.h`', la gestione effettiva dell'array di stringhe a cui punta ***environ*** è inserita nei file contenuti nella directory '`lib/stdlib/`' di os16. Per la precisione, nel file '`lib/stdlib/environment.c`' si dichiara l'array di caratteri ***_environment_table[][]*** e array di puntatori a caratteri ***_environment[]***:

```
char  _environment_table[ARG_MAX/32][ARG_MAX/16];
char *_environment[ARG_MAX/32+1];
```

L'array ***_environment_table[][]*** viene inizializzato con lo stato delle variabili di ambiente ereditate con l'avvio del processo; inoltre, all'array ***_environment[]*** vengono assegnati i puntatori alle varie stringhe che si possono estrapolare da ***_environment_table[]***. Questo lavoro iniziale avviene per opera della funzione ***_environment_setup()***, la quale viene avviata a sua volta dal file '`crt0.s`'. Successivamente, nello stesso file '`crt0.s`', viene copiato l'indirizzo dell'***_environment[]*** nella variabile ***environ***, di cui sopra.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]
'`lib/stdlib.h`' [u0.10]
'`lib/unistd/environ.c`' [i161.17.8]
'`applic/crt0.s`' [i162.1.9]
'`lib/stdlib/environment.c`' [i161.10.9]
'`lib/stdlib/getenv.c`' [i161.10.11]
'`lib/stdlib/putenv.c`' [i161.10.14]
'`lib/stdlib/setenv.c`' [i161.10.17]
'`lib/stdlib/unsetenv.c`' [i161.10.20]

VEDERE ANCHE

getenv(3) [u0.51], *putenv(3)* [u0.82], *setenv(3)* [u0.94], *unsetenv(3)* [u0.94].

os16: undocumented(7)

Questa sezione ha il solo scopo di raccogliere i riferimenti ipertestuali dei listati che, per qualche ragione, sono privi di una documentazione specifica.

Sezione 8: comandi per l'amministrazione del sistema

os16: getty(8) 1503
os16: init(8) 1503
os16: MAKEDEV(8) 1504
os16: mount(8) 1504
os16: umount(8) 1505

`getty` 1503 `init` 1503 `MAKEDEV` 1504 `mount` 1504 `umount` 1504

os16: getty(8)

NOME

'**`getty`**' - predisposizione di un terminale e avvio di '**`login`**'

SINTASSI

```
getty terminale
```

DESCRIZIONE

Il programma '**`getty`**' viene avviato da '**`init`**' per predisporre il terminale, ripristinando anche i permessi predefiniti, e per avviare successivamente il programma '**`login`**'. Prima di avviare '**`login`**', '**`getty`**' visualizza il contenuto del file '`/etc/issue`', se disponibile, inoltre mostra almeno l'indicazione del terminale attuale. Va osservato che questa realizzazione di '**`getty`**' lascia a '**`login`**' il compito di chiedere l'inserimento del nominativo utente.

FILE

'`/etc/issue`'
'**`getty`**' visualizza il contenuto di questo file prima di avviare '**`login`**'.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]
'`applic/getty.c`' [i162.1.12]

VEDERE ANCHE

login(1) [u0.12], *issue(5)* [u0.2].

os16: init(8)

NOME

'**`init`**' - progenitore di tutti gli altri processi

SINTASSI

```
init
```

DESCRIZIONE

Il programma '**`init`**' viene avviato dal kernel (deve trattarsi precisamente del file '`/bin/init`') come primo e unico processo figlio del kernel stesso. Pertanto, '**`init`**' deve assumere il numero PID uno.

Questa realizzazione di '**`init`**' si limita a leggere il file '`/etc/inittab`' per determinare quali programmi figli avviare, senza poter distinguere da diversi livelli di esecuzione. In pratica, all'interno di questo file si indica l'uso di '**`getty`**', per la gestione dei terminali disponibili.

FILE

'`/etc/inittab`'
Contiene l'indicazione dei processi che '**`init`**' deve avviare.

DIFETTI

Con os16 non è possibile associare ai segnali un'azione diversa da quella predefinita; quindi '**`init`**' non può essere informato dell'intenzione di arrestare il sistema. Pertanto, tale funzionalità non è stata realizzata nella versione di '**`init`**' di os16.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`applic/init.c`' [i162.1.13]

VEDERE ANCHE

inittab(5) [u0.1].

os16: MAKEDEV(8)

NOME

'**`MAKEDEV`**' - creazione dei file di dispositivo

SINTASSI

```
MAKEDEV
```

DESCRIZIONE

'**`MAKEDEV`**' è un programma che crea, nella directory corrente, tutti i file di dispositivo previsti per os16. Tali file devono trovarsi normalmente nella directory '`/dev/`', pertanto, prima di usare '**`MAKEDEV`**' è necessario che la directory corrente corrisponda precisamente a tale posizione.

OPZIONI

Non sono previste opzioni per l'uso di '**`MAKEDEV`**', dal momento che vengono creati tutti i file di dispositivo, considerato il loro numero estremamente limitato.

NOTE

Tradizionalmente '**`MAKEDEV`**' viene realizzato in forma di script, ma os16 non dispone di una shell adeguata e non è possibile utilizzare script.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`lib/sys/os16.h`' [u0.12]

'`applic/MAKEDEV.c`' [i162.1.1]

os16: mount(8)

NOME

'**`mount`**', '**`umount`**' - innesto e distacco di un file system

SINTASSI

```
mount dispositivo dir_innesto [opzioni]
```

```
umount directory
```

DESCRIZIONE

'**`mount`**' innesta il file system contenuto nell'unità di memorizzazione rappresentata dal file di dispositivo che va indicato come primo argomento, nella directory che appare come secondo argomento. Eventualmente si possono specificare delle opzioni di innesto, come terzo argomento.

'**`umount`**' stacca il file system innestato precedentemente nella directory indicata come unico argomento del comando.

OPZIONI DI INNESTO

Opzione	Descrizione
`ro`	Innesta il file system in sola lettura.
`rw`	Innesta il file system in lettura e scrittura. Si tratta comunque del comportamento predefinito, in mancanza di un'opzione contraria.

DIFETTI

Non viene preso in considerazione un eventuale file '`/etc/fstab`'; inoltre, l'utente non può conoscere lo stato degli innesti già in essere e, a questo proposito, l'uso di '**`mount`**' senza argomenti produce semplicemente un errore.

FILE SORGENTI

'`applic/crt0.s`' [i162.1.9]

'`applic/mount.c`' [i162.1.21]

'`applic/umount.c`' [i162.1.27]

os16: umount(8)

Vedere *mount(8)* [u0.4].

os16: devices(9) 1509
os16: dev_io(9) 1511
os16: dev_dsk(9) 1511
os16: dev_kmem(9) 1512
os16: dev_mem(9) 1513
os16: dev_tty(9) 1513
os16: diag(9) 1514
os16: fs(9) 1514
os16: fd_chmod(9) 1516
os16: fd_chown(9) 1517
os16: fd_close(9) 1518
os16: fd_dup(9) 1518
os16: fd_dup2(9) 1520
os16: fd_fcntl(9) 1520
os16: fd_lseek(9) 1521
os16: fd_open(9) 1522
os16: fd_read(9) 1524
os16: fd_reference(9) 1525
os16: fd_stat(9) 1526
os16: fd_write(9) 1526
os16: file_reference(9) 1527
os16: file_stdio_dev_make(9) 1527
os16: inode_alloc(9) 1528
os16: inode_check(9) 1529
os16: inode_dir_empty(9) 1530
os16: inode_file_read(9) 1530
os16: inode_file_write(9) 1531
os16: inode_free(9) 1532
os16: inode_fzones_read(9) 1532
os16: inode_fzones_write(9) 1533
os16: inode_get(9) 1533
os16: inode_put(9) 1534
os16: inode_reference(9) 1535
os16: inode_save(9) 1535
os16: inode_stdio_dev_make(9) 1536
os16: inode_truncate(9) 1537
os16: inode_zone(9) 1537
os16: path_chdir(9) 1538
os16: path_chmod(9) 1539
os16: path_chown(9) 1540
os16: path_device(9) 1541
os16: path_fix(9) 1541
os16: path_full(9) 1542
os16: path_inode(9) 1542
os16: path_inode_link(9) 1543
os16: path_link(9) 1544
os16: path_mkdir(9) 1545
os16: path_mknod(9) 1546
os16: path_mount(9) 1547
os16: path_stat(9) 1548
os16: path_umount(9) 1548
os16: path_unlink(9) 1548
os16: sb_inode_status(9) 1549
os16: sb_mount(9) 1550
os16: sb_reference(9) 1551
os16: sb_save(9) 1552

os16: sb_zone_status(9) 1552
os16: stat(9) 1552
os16: zone_alloc(9) 1554
os16: zone_free(9) 1555
os16: zone_read(9) 1555
os16: ibm_i86(9) 1556
os16: k_libc(9) 1560
os16: main(9) 1560
os16: memory(9) 1560
os16: proc(9) 1561
os16: isr_1C(9) 1564
os16: ivt_load(9) 1565
os16: proc_available(9) 1566
os16: proc_dump_memory(9) 1566
os16: proc_find(9) 1567
os16: proc_init(9) 1567
os16: proc_reference(9) 1568
os16: proc_sch_signals(9) 1568
os16: proc_sch_terminals(9) 1569
os16: proc_sch_timers(9) 1569
os16: proc_scheduler(9) 1570
os16: proc_sig_chld(9) 1571
os16: proc_sig_cont(9) 1571
os16: proc_sig_core(9) 1572
os16: proc_sig_ignore(9) 1573
os16: proc_sig_on(9) 1573
os16: proc_sig_status(9) 1574
os16: proc_sig_stop(9) 1575
os16: proc_sig_term(9) 1575
os16: proc_sys_exec(9) 1575
os16: proc_sys_exit(9) 1577
os16: proc_sys_fork(9) 1578
os16: proc_sys_kill(9) 1579
os16: proc_sys_seteuid(9) 1580
os16: proc_sys_setuid(9) 1581
os16: proc_sys_signal(9) 1582
os16: proc_sys_wait(9) 1583
os16: sysroutine(9) 1584
os16: tty(9) 1585

devices.h 1509 dev_dsk() 1511 dev_io() 1511 dev_kmem() 1512 dev_mem() 1513 dev_tty() 1513 diag.h 1514 fd_chmod() 1516 fd_chown() 1517 fd_close() 1518 fd_dup() 1518 fd_dup2() 1518 fd_fcntl() 1520 fd_lseek() 1521 fd_open() 1522 fd_read() 1524 fd_reference() 1525 fd_stat() 1552 fd_write() 1526 file_reference() 1527 file_stdio_dev_make() 1527 fs.h 1514 ibm_i86.h 1556 inode_alloc() 1528 inode_check() 1529 inode_dir_empty() 1530 inode_file_read() 1530 inode_file_write() 1531 inode_free() 1532 inode_fzones_read() 1532 inode_fzones_write() 1532 inode_get() 1533 inode_put() 1534 inode_reference() 1535 inode_save() 1535 inode_stdio_dev_make() 1536 inode_truncate() 1537 inode_zone() 1537 isr_1C 1564 isr_80 1564 ivt_load() 1565 k_libc.h 1560 main.h 1560 memory.h 1560 path_chdir() 1538 path_chmod() 1539 path_chown() 1540 path_device() 1541 path_fix() 1541 path_full() 1542 path_inode() 1542 path_inode_link() 1543 path_link() 1544 path_mkdir() 1545 path_mknod() 1546 path_mount() 1547 path_stat() 1552 path_umount() 1547 path_unlink() 1548 proc.h 1561 proc_available() 1566 proc_dump_memory() 1566 proc_find() 1567 proc_init() 1567 proc_reference() 1568 proc_scheduler() 1570 proc_sch_signals() 1568 proc_sch_terminals() 1569 proc_sch_timers() 1569 proc_sig_chld() 1571 proc_sig_cont() 1571 proc_sig_core() 1572 proc_sig_ignore() 1573 proc_sig_off() 1573 proc_sig_on() 1573 proc_sig_status() 1574 proc_sig_stop() 1575 proc_sig_term() 1575 proc_sys_exec() 1575 proc_sys_exit() 1577 proc_sys_fork() 1578 proc_sys_kill() 1579 proc_sys_seteuid() 1580 proc_sys_setuid() 1581 proc_sys_signal() 1582 proc_sys_wait() 1583 sb_inode_status() 1549 sb_mount() 1550 sb_reference() 1551 sb_save() 1552 sb_zone_status() 1549 sysroutine() 1584 tty.h 1585 zone_alloc() 1554 zone_free() 1554 zone_read() 1555 zone_write() 1555 _ivt_load() 1565

os16: devices(9)

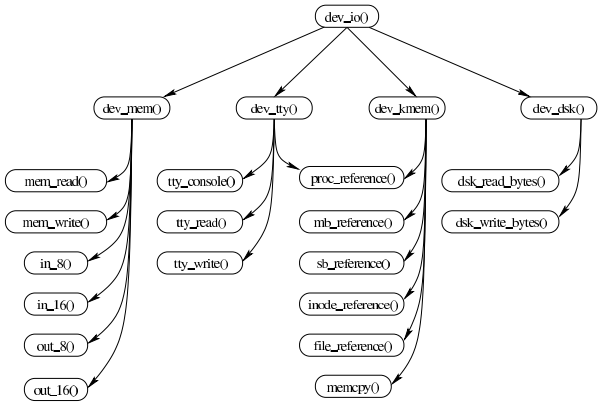
Il file ‘kernel/devices.h’ [u0.2] descrive ciò che serve per la gestione dei dispositivi. Tuttavia, la definizione dei numeri di dispositivo è contenuta nel file ‘lib/sys/os16.h’ [u0.12], il quale viene incluso da ‘devices.h’.

Tabella u147.4. Classificazione dei dispositivi di os16.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
DEV_MEM	r/w	diretto	Permette l’accesso alla memoria, in modo indiscriminato, perché os16 non offre alcun tipo di protezione al riguardo.
DEV_NULL	r/w	nessuno	Consente la lettura e la scrittura, ma non si legge e non si scrive alcunché.
DEV_PORT	r/w	sequenziale	Consente di leggere e scrivere da o verso una porta di I/O, individuata attraverso l’indirizzo di accesso (l’indirizzo, o meglio lo scostamento, viene trattato come la porta a cui si vuole accedere). Tuttavia, la dimensione dell’informazione da trasferire è valida solo se si tratta di uno o di due byte: per la dimensione di un byte si usano le funzioni ***in_8()*** e ***out_8()***; per due byte si usano le funzioni ***in_16()*** e ***out_16()***. Per dimensioni differenti la lettura o la scrittura non ha effetto.
DEV_ZERO	r	sequenziale	Consente solo la lettura di valori a zero (zero inteso in senso binario).
DEV_TTY	r/w	sequenziale	Rappresenta il terminale virtuale del processo attivo.
DEV_DSK***n***	r/w	diretto	Rappresenta l’unità a dischi ***n***. os16 non gestisce le partizioni.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_PS`	r	diretto	Rappresenta la tabella contenente le informazioni sui processi. L'indirizzo di accesso indica il numero del processo di partenza; la dimensione da leggere dovrebbe essere abbastanza grande da contenere un processo, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_MMP`	r	sequenziale	Rappresenta la mappa della memoria, alla quale si può accedere solo dal suo principio. In pratica, l'indirizzo di accesso viene ignorato, mentre conta solo la quantità di byte richiesta.
`DEV_KMEM_SB`	r	diretto	Rappresenta la tabella dei super blocchi (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il super blocco; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un super blocco, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_INODE`	r	diretto	Rappresenta la tabella degli inode (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare l'inode; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un inode, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_FILE`	r	diretto	Rappresenta la tabella dei file (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il file; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di un file, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_CONSOLE`	r/w	sequenziale	Legge o scrive relativamente alla console attiva la quantità di byte richiesta, ignorando l'indirizzo di accesso.
`DEV_CONSOLE`***n***	r/w	sequenziale	Legge o scrive relativamente alla console ***n*** la quantità di byte richiesta, ignorando l'indirizzo di accesso.

Figura u147.1. Interdipendenza tra la funzione ***dev_io()*** e le altre. I collegamenti con le funzioni ***major()*** e ***minor()*** sono omesse.



os16: dev_io(9) «

NOME

'**`dev_io`**' - interfaccia di accesso ai dispositivi

SINTASSI

```
<kernel/devices.h>
ssize_t dev_io (pid_t pid, dev_t device, int rw, off_t offset,
                void *buffer, size_t size, int *eof);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`dev_t` ***device***	Dispositivo, in forma numerica.
`int` ***rw***	Può assumere i valori '**`DEV_READ`**' o '**`DEV_WRITE`**', per richiedere rispettivamente un accesso in lettura oppure in scrittura.
`off_t` ***offset***	Posizione per l'accesso al dispositivo.
`void *`***buffer***	Memoria tampone, per la lettura o la scrittura.
`size_t` ***size***	Quantità di byte da leggere o da scrivere.
`int *`***eof***	Puntatore a una variabile in cui annotare, eventualmente, il raggiungimento della fine del file.

DESCRIZIONE

La funzione ***dev_io()*** è un'interfaccia generale per l'accesso ai dispositivi gestiti da os16.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti o scritti effettivamente. In caso di errore restituisce il valore `-1` e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENODEV`	Il numero del dispositivo non è valido.
`EIO`	Errore di input-output.

FILE SORGENTI

'`kernel/devices.h`' [u0.2]

'`kernel/devices/dev_io.c`' [i160.2.2]

VEDERE ANCHE

dev_dsk(9) [i159.1.2], *dev_kmem(9)* [i159.1.3], *dev_mem(9)* [i159.1.4], *dev_tty(9)* [i159.1.5].

NOME

‘**dev_dsk**’ - interfaccia di accesso alle unità di memorizzazione di massa

SINTASSI

```
<kernel/devices.h>
ssize_t dev_dsk (pid_t pid, dev_t device, int rw,
                 off_t offset, void *buffer,
                 size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_dsk()*** consente di accedere alle unità di memorizzazione di massa, che per os16 si riducono ai soli dischetti da 1440 Kibyte.

Per il significato degli argomenti, il valore restituito e gli eventuali errori, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

‘`kernel/devices.h`’ [u0.2]

‘`kernel/devices/dev_io.c`’ [i160.2.2]

‘`kernel/devices/dev_dsk.c`’ [i160.2.1]

NOME

‘**dev_kmem**’ - interfaccia di accesso alle tabelle di dati del kernel, rappresentate in memoria

SINTASSI

```
<kernel/devices.h>
ssize_t dev_kmem (pid_t pid, dev_t device, int rw,
                  off_t offset,
                  void *buffer, size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_kmem()*** consente di accedere, solo in lettura, alle porzioni di memoria che il kernel utilizza per rappresentare alcune tabelle importanti. Per poter interpretare ciò che si ottiene occorre riprodurre la struttura di un elemento della tabella a cui si è interessati, pertanto occorre incorporare il file di intestazione del kernel che la descrive.

Dispositivo	Tabella a cui si riferisce
`DEV_KMEM_PS`	Si accede alla tabella dei processi, all’elemento rappresentato da ***offset***: ***proc_table[offset]***.
`DEV_KMEM_MMP`	Si accede alla mappa della memoria, ovvero la tabella ***mb_table[]***, senza considerare il valore di ***offset***.
`DEV_KMEM_SB`	Si accede alla tabella dei super blocchi, all’elemento rappresentato da ***offset***: ***sb_table[offset]***.
`DEV_KMEM_INODE`	Si accede alla tabella degli inode, all’elemento rappresentato da ***offset***: ***inode_table[offset]***.
`DEV_KMEM_FILE`	Si accede alla tabella dei file di sistema, all’elemento rappresentato da ***offset***: ***file_table[offset]***.

Per il significato degli argomenti della chiamata, per interpretare il valore restituito e gli eventuali errori, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

‘`kernel/devices.h`’ [u0.2]

‘`kernel/devices/dev_io.c`’ [i160.2.2]

‘`kernel/devices/dev_kmem.c`’ [i160.2.3]

NOME

‘**dev_mem**’ - interfaccia di accesso alla memoria, in modo indiscriminato

SINTASSI

```
<kernel/devices.h>
ssize_t dev_mem (pid_t pid, dev_t device, int rw,
                 off_t offset, void *buffer,
                 size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_mem()*** consente di accedere, in lettura e in scrittura alla memoria e alle porte di input-output.

Dispositivo	Descrizione
`DEV_MEM`	Si tratta della memoria centrale, complessiva, dove il valore di ***offset*** rappresenta l’indirizzo efficace (complessivo) a partire da zero.
`DEV_NULL`	Si tratta di ciò che realizza il file di dispositivo tradizionale ‘`/dev/null`’: la scrittura si perde semplicemente e la lettura non dà alcunché.
`DEV_ZERO`	Si tratta di ciò che realizza il file di dispositivo tradizionale ‘`/dev/zero`’: la scrittura si perde semplicemente e la lettura produce byte a zero (zero binario).
`DEV_PORT`	Consente l’accesso alle porte di input-output. La dimensione rappresentata da ***size*** può essere solo pari a uno o due: una dimensione pari a uno richiede di comunicare un solo byte con una certa porta; una dimensione pari a due richiede la comunicazione di un valore a 16 bit. Il valore di ***offset*** serve a individuare la porta di input-output con cui si intende comunicare (leggere o scrivere un valore).

Per quanto non viene descritto qui, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

‘`kernel/devices.h`’ [u0.2]

‘`kernel/devices/dev_io.c`’ [i160.2.2]

‘`kernel/devices/dev_mem.c`’ [i160.2.4]

NOME

‘**dev_tty**’ - interfaccia di accesso alla console

SINTASSI

```
<kernel/devices.h>
ssize_t dev_tty (pid_t pid, dev_t device, int rw, off_t offset,
                 void *buffer, size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_tty()*** consente di accedere, in lettura e in scrittura, a una console virtuale, scelta in base al numero del dispositivo.

Quando la lettura richiede l’attesa per l’inserimento da tastiera, se il processo elaborativo ***pid*** non è il kernel, allora viene messo in pausa, in attesa di un evento legato al terminale.

Il sistema di gestione del terminale è molto povero con os16. Va osservato che il testo letto viene anche visualizzato automaticamente. Quando un processo non vuole mostrare il testo sullo schermo, deve provvedere a sovrascriverlo immediatamente, facendo arretrare il cursore preventivamente.

Per quanto non viene descritto qui, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

‘`kernel/devices.h`’ [u0.2]

‘`kernel/devices/dev_io.c`’ [i160.2.2]

‘`kernel/devices/dev_tty.c`’ [i160.2.5]

os16: diag(9)

«

Il file ‘`kernel/diag.h`’ [u0.3] descrive alcune funzioni e macroistruzioni, per uso diagnostico. Lo scopo di queste è di mostrare o di rendere visualizzabile alcune informazioni interne alla gestione del kernel.

Alcune di queste funzioni sono usate, altre no. Per esempio durante il funzionamento interattivo del kernel vengono usate ***print_proc_list()***, ***print_segments()***, ***print_kmem()***, ***print_time()*** e ***print_mb_map()***.

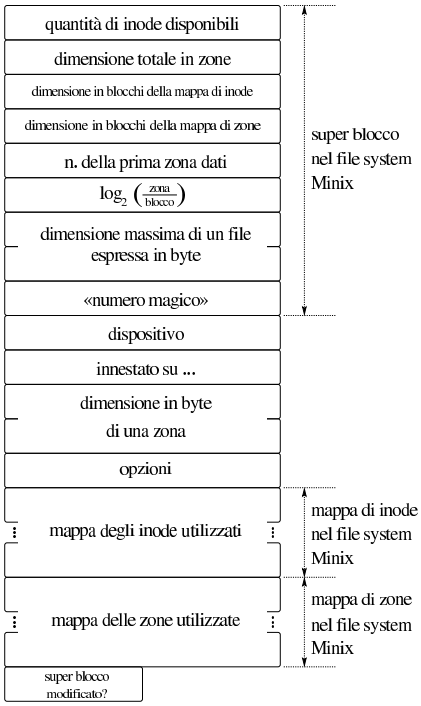
os16: fs(9)

«

Il file ‘`kernel/fs.h`’ [u0.4] descrive ciò che serve per la gestione del file system, che per os16 corrisponde al tipo Minix 1.

La gestione del file system, a livello complessivo di sistema, è suddivisa in tre aspetti principali: super blocco, inode e file. Per ognuno di questi è prevista una tabella (di super blocchi, di inode e di file). Seguono delle figure che descrivono l’organizzazione di queste tabelle.

Figura u148.1. Struttura del tipo ‘**`sb_t`**’, corrispondente agli elementi dell’array ***sb_table[]***.

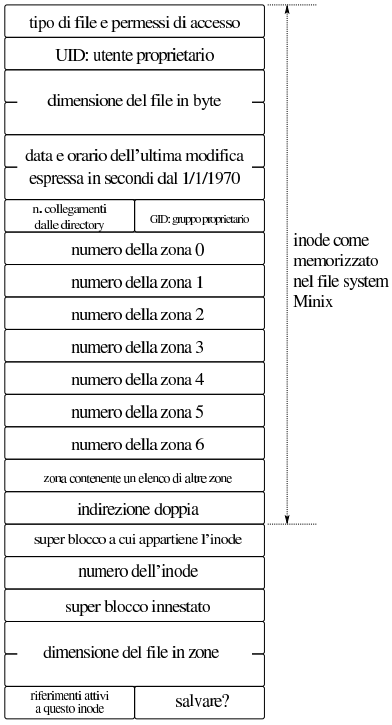


Listato u148.2. Struttura del tipo ‘**`sb_t`**’, corrispondente agli elementi dell’array ***sb_table[]***.

```
typedef struct sb        sb_t;

struct sb {
  uint16_t  inodes;
  uint16_t  zones;
  uint16_t  map_inode_blocks;
  uint16_t  map_zone_blocks;
  uint16_t  first_data_zone;
  uint16_t  log2_size_zone;
  uint32_t  max_file_size;
  uint16_t  magic_number;
  //------------------------
  dev_t     device;
  inode_t  *inode_mounted_on;
  blksize_t blksize;
  int       options;
  uint16_t  map_inode[SB_MAP_INODE_SIZE];
  uint16_t  map_zone[SB_MAP_ZONE_SIZE];
  char      changed;
};
```

Figura u148.6. Struttura del tipo ‘**`inode_t`**’, corrispondente agli elementi dell’array ***inode_table[]***.

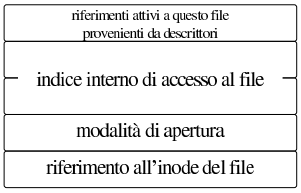


Listato u148.7. Struttura del tipo '**inode_t**', corrispondente agli elementi dell'array ***inode_table[]***.

```
<verbatimpre width="60">
<![CDATA[
typedef struct inode     inode_t;

struct inode {
    mode_t        mode;
    uid_t         uid;
    ssize_t       size;
    time_t        time;
    uint8_t       gid;
    uint8_t       links;
    zno_t         direct[7];
    zno_t         indirect1;
    zno_t         indirect2;
    //------------------------
    sb_t         *sb;
    ino_t         ino;
    sb_t         *sb_attached;
    blkcnt_t      blkcnt;
    unsigned char references;
    char          changed;
};
```

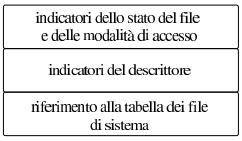
Figura u148.13. Struttura del tipo '**file_t**', corrispondente agli elementi dell'array ***file_table[]***.




```
typedef struct file file_t;

struct file {
  int      references;
  off_t    offset;
  int      oflags;
  inode_t *inode;
};
```

Figura u148.16. Struttura del tipo '**fd_t**', con cui si costituiscono gli elementi delle tabelle dei descrittori di file, una per ogni processo.




```
typedef struct fd fd_t;
struct fd {
    int     fl_flags;
    int     fd_flags;
    file_t *file;
};
```

os16: fd_chmod(9)

«

NOME

'**fd_chmod**' - cambiamento della modalità dei permessi di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_chmod (pid_t pid, int fdn, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.
`mode_t` ***mode***	Modalità dei permessi, di cui si prendono in considerazione solo i 12 bit meno significativi.

DESCRIZIONE

La funzione ***fs_chmod()*** cambia la modalità dei permessi del file aperto con il descrittore numero ***fdn***, secondo il valore contenuto nel parametro ***mode***, di cui però si considerano solo gli ultimi 12 bit. L'operazione viene svolta per conto del processo ***pid***, il quale deve avere i privilegi necessari per poter intervenire così. La modifica della modalità dei permessi raggiunge l'inode del file a cui fa capo il descrittore in questione; pertanto l'inode viene necessariamente salvato dopo la modifica. Il fatto che il descrittore di file possa essere stato aperto in sola lettura, non impedisce la modifica dell'inode attuata da questa funzione.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_FCHMOD**'. Nella libreria standard, si avvale di questa funzionalità *fchmod(2)* [u0.4].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file ***fdn*** non è valido.
`EACCES`	Il processo elaborativo non ha i privilegi necessari nei confronti del file.

FILE SORGENTI

'`lib/sys/stat/fchmod.c`' [i161.13.2]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_chmod.c`' [i160.4.1]

VEDERE ANCHE

fchmod(2) [u0.4], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_chown(9)

«

NOME

'**fd_chown**' - cambiamento della proprietà di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_chown (pid_t pid, int fdn, uid_t uid, gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.
`uid_t` ***uid***	Nuova proprietà che si intende attribuire al file.
`gid_t` ***gid***	Nuovo gruppo che si intenderebbe attribuire al file.

DESCRIZIONE

La funzione ***fs_chown()*** cambia la proprietà del file già aperto, individuato attraverso il suo descrittore. L'operazione viene svolta per conto del processo ***pid***, il quale deve avere i privilegi necessari per poter intervenire così: in pratica deve trattarsi di un processo con identità efficace pari a zero, perché os16 non considera la gestione dei gruppi. La modifica della proprietà raggiunge l'inode del file a cui fa capo il descrittore in questione; pertanto l'inode viene necessariamente salvato dopo la modifica. Il fatto che il descrittore di file possa essere stato aperto in sola lettura, non impedisce la modifica dell'inode attuata da questa funzione.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_FCHOWN**'. Nella libreria standard, si avvale di questa funzionalità *fchown(2)* [u0.4].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file ***fdn*** non è valido.
`EACCES`	Il processo elaborativo non ha i privilegi necessari nei confronti del file.

FILE SORGENTI

‘`lib/unistd/fchown.c`’ [i161.17.16]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_chown.c`’ [i160.4.2]

VEDERE ANCHE

fchown(2) [u0.5], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_close(9)

NOME

‘**`fd_close`**’ - chiusura di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_close (pid_t pid, int fdn);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.

DESCRIZIONE

La funzione ***fd_close()*** chiude il descrittore di file specificato come argomento. Per ottenere questo risultato, oltre che intervenire nella tabella dei descrittori associata al processo elaborativo specificato come argomento, riduce il contatore dei riferimenti nella voce corrispondente della tabella dei file; se però questo contatore raggiunge lo zero, anche l’inode viene liberato, attraverso *inode_put(9)* [i159.3.24].

Questa funzione viene usata in modo particolare da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_CLOSE`**’. Nella libreria standard, si avvale di questa funzionalità *close(2)* [u0.7]. La funzione ‘**`fd_close`**’ è comunque usata internamente al kernel, in tutte le occasioni in cui la chiusura di un descrittore deve avvenire in modo implicito.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
`-1`	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file ***fdn*** non è valido.

FILE SORGENTI

‘`lib/unistd/close.c`’ [i161.17.5]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_close.c`’ [i160.4.3]

VEDERE ANCHE

close(2) [u0.7], *sysroutine(9)* [i159.8.28], *inode_put(9)* [i159.3.24].

os16: fd_dup(9)

NOME

‘**`fd_dup`**’, ‘**`fd_dup2`**’ - duplicazione di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_dup  (pid_t pid, int fdn_old, int fdn_min);
int fd_dup2 (pid_t pid, int fdn_old, int fdn_new);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn_old***	Numero del descrittore di file da duplicare.
`int` ***fdn_min***	Primo numero di descrittore da usare per la copia.
`int` ***fdn_new***	Numero di descrittore da usare per la copia.

DESCRIZIONE

Le funzioni ***fd_dup()*** e ***fd_dup2()*** duplicano un descrittore, nel senso che sdoppiano l’accesso a un file in due descrittori. La funzione ***fd_dup()***, per il duplicato da realizzare, cerca un descrittore libero, cominciando da ***fdn_min*** e continuando progressivamente, fino al primo disponibile. La funzione ***fd_dup2()***, invece, richiede di specificare esattamente il descrittore da usare per il duplicato, con la differenza che, se ***fdn_new*** è già utilizzato, prima della duplicazione viene chiuso.

In entrambi i casi, il descrittore ottenuto dalla copia, viene privato dell’indicatore ‘**`FD_CLOEXEC`**’, ammesso che nel descrittore originale ci fosse.

Queste funzioni vengono usate da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_DUP`**’ e ‘**`SYS_DUP2`**’. Inoltre, la funzione *fd_fcntl(9)* [i159.3.6] si avvale di ***fd_dup()*** per la duplicazione di un descrittore. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***fd_dup()*** e ***fd_dup2()*** sono *dup(2)* [u0.8] e *dup2(2)* [u0.8].

VALORE RESTITUITO

Le due funzioni restituiscono il numero del descrittore prodotto dalla duplicazione. In caso di errore, invece, restituiscono il valore `-1`, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore di ***fdn_min*** o ***fdn_new*** è impossibile.
`EBADF`	Il valore di ***fdn_old*** non è valido.
`EMFILE`	Non è possibile allocare un nuovo descrittore.

FILE SORGENTI

‘`lib/unistd/dup.c`’ [i161.17.6]

‘`lib/unistd/dup2.c`’ [i161.17.7]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_dup.c`’ [i160.4.4]

‘`kernel/fs/fd_dup2.c`’ [i160.4.5]

VEDERE ANCHE

dup(2) [u0.8], *dup2(2)* [u0.8], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_dup2(9)

Vedere *fd_dup(9)* [i159.3.4].

os16: fd_fcntl(9)

NOME

'**fd_fcntl**' - configurazione e intervento sui descrittori di file

SINTASSI

```
<kernel/fs.h>
int fd_fcntl (pid_t pid, int fdn, int cmd, int arg);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore a cui si fa riferimento.
`int` ***cmd***	Comando di ***fd_fcntl()***.
`int` ***arg***	Argomento eventuale del comando.

DESCRIZIONE

La funzione ***fd_fcntl()*** esegue un'operazione, definita dal parametro ***cmd***, sul descrittore ***fdn***. A seconda del tipo di operazione richiesta, può essere preso in considerazione anche l'argomento corrispondente al parametro ***arg***. Il valore del parametro ***cmd*** che rappresenta l'operazione richiesta, va fornito in forma di costante simbolica, come descritto nell'elenco seguente. Tali macro-variabili derivano dalle dichiarazioni contenute nel file '`lib/sys/fcntl.h`'.

Sintassi	Descrizione
`fd_fcntl (`***pid***`, `***fdn***`, F_DUPFD, (int) `***fdn_min***`)`	Richiede la duplicazione del descrittore di file ***fdn***, in modo tale che la copia abbia il numero di descrittore minore possibile, ma maggiore o uguale a quello indicato come argomento ***fdn_min***.
`fd_fcntl (`***pid***`, `***fdn***`, F_GETFD, 0)` `fd_fcntl (`***pid***`, `***fdn***`, F_SETFD, (int) `***fd_flags***`)`	Rispettivamente, legge o imposta, gli indicatori del descrittore di file ***fdn*** (eventualmente noti come *fd flags*). È possibile impostare un solo indicatore, '**FD_CLOEXEC**', pertanto, al posto di ***fd_flags*** si può mettere solo la costante '**FD_CLOEXEC**'.
`fd_fcntl (`***pid***`, `***fdn***`, F_GETFL, 0)` `fd_fcntl (`***pid***`, `***fdn***`, F_SETFL, (int) `***fl_flags***`)`	Rispettivamente, legge o imposta, gli indicatori dello stato del file, relativi al descrittore ***fdn***. Per impostare questi indicatori, vanno combinate delle costanti simboliche: '**O_RDONLY**', '**O_WRONLY**', '**O_RDWR**', '**O_CREAT**', '**O_EXCL**', '**O_NOCTTY**', '**O_TRUNC**'.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_FCNTL**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_fcntl()*** è *fcntl(2)* [u0.13].

VALORE RESTITUITO

Il significato del valore restituito dalla funzione dipende dal tipo di operazione richiesta, come sintetizzato dalla tabella successiva.

Operazione richiesta	Significato del valore restituito
`F_DUPFD`	Si ottiene il numero del descrittore prodotto dalla copia, oppure `-1` in caso di errore.
`F_GETFD`	Si ottiene il valore degli indicatori del descrittore (*fd flags*, oppure `-1` in caso di errore.
`F_GETFL`	Si ottiene il valore degli indicatori del file (*fl flags*, oppure `-1` in caso di errore.

`F_GETOWN`
`F_SETOWN`
`F_GETLK`
`F_SETLK`
`F_SETLKW`
	Si ottiene `-1`, in quanto si tratta di operazioni non realizzate in questa versione della funzione, per os16.

altri tipi di operazione	Si ottiene 0 in caso di successo, oppure `-1` in caso di errore.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file non è valido.
`EINVAL`	È stato richiesto un tipo di operazione non valido.
`EMFILE`	Non è possibile duplicare il descrittore, perché non ce ne sono di liberi.

FILE SORGENTI

'`lib/fcntl/fcntl.c`' [i161.4.2]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_fcntl.c`' [i160.4.6]

VEDERE ANCHE

fcntl(2) [u0.13], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *fd_dup(9)* [i159.3.4].

os16: fd_lseek(9)

NOME

'**fd_lseek**' - riposizionamento dell'indice di accesso a un descrittore di file

SINTASSI

```
<kernel/fs.h>
off_t fd_lseek (pid_t pid, int fdn, off_t offset, int whence);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore a cui si fa riferimento.
`int` ***cmd***	Comando di ***fd_fcntl()***.
`off_t` ***offset***	Scostamento, positivo o negativo, a partire dalla posizione indicata da ***whence***.
`int` ***whence***	Punto di riferimento iniziale a cui applicare lo scostamento.

DESCRIZIONE

La funzione ***fd_lseek()*** consente di riposizionare l'indice di accesso interno al descrittore di file ***fdn***. Per fare questo occorre prima determinare un punto di riferimento, rappresentato dal parametro ***whence***, dove va usata una macro-variabile definita nel file '`lib/unistd.h`'. Può trattarsi dei casi seguenti.

Valore di ***whence***	Significato
`SEEK_SET`	lo scostamento si riferisce all'inizio del file.
`SEEK_CUR`	lo scostamento si riferisce alla posizione che ha già l'indice interno al file.
`SEEK_END`	lo scostamento si riferisce alla fine del file.

Lo scostamento indicato dal parametro ***offset*** si applica a partire dalla posizione a cui si riferisce ***whence***, pertanto può avere segno positivo o negativo, ma in ogni caso non è possibile collocare l'indice prima dell'inizio del file.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_LSEEK`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_lseek()*** è *lseek(2)* [u0.24].

VALORE RESTITUITO

Se l'operazione avviene con successo, la funzione restituisce il valore dell'indice riposizionato, preso come scostamento a partire dall'inizio del file. In caso di errore, restituisce invece il valore `-1`, aggiornando di conseguenza anche la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il valore di ***whence*** non è contemplato, oppure la combinazione tra ***whence*** e ***offset*** non è valida.

FILE SORGENTI

'`lib/unistd/lseek.c`' [i161.17.27]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_lseek.c`' [i160.4.7]

VEDERE ANCHE

lseek(2) [u0.24], *sysroutine(9)* [i159.8.28], *fd_reference(9)* [i159.3.10].

os16: fd_open(9)

«

NOME

'**`fd_open`**' - apertura di un file puro e semplice oppure di un file di dispositivo

SINTASSI

```
<kernel/fs.h>
int fd_open (pid_t pid, const char *path, int oflags,
             mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`char *`***path***	Il percorso assoluto del file da aprire.
`int` ***oflags***	Opzioni di apertura.
`mode_t` ***mode***	Tipo di file e modalità dei permessi, nel caso il file debba essere creato contestualmente.

DESCRIZIONE

La funzione ***fd_open()*** apre un file, indicato attraverso il percorso ***path***, in base alle opzioni rappresentate dagli indicatori ***oflags***. A seconda del tipo di indicatori specificati, il parametro ***mode*** potrebbe essere preso in considerazione.

Quando la funzione porta a termine correttamente il proprio compito, restituisce il numero del descrittore del file associato, il quale è sempre quello di valore più basso disponibile per il processo elaborativo a cui ci si riferisce.

Il parametro ***oflags*** richiede necessariamente la specificazione della modalità di accesso, attraverso la combinazione appropriata dei valori: '**`O_RDONLY`**', '**`O_WRONLY`**', '**`O_RDWR`**'. Inoltre, si possono combinare altri indicatori: '**`O_CREAT`**', '**`O_TRUNC`**', '**`O_APPEND`**'.

Opzione	Descrizione
`O_RDONLY`	Richiede un accesso in lettura.
`O_WRONLY`	Richiede un accesso in scrittura.
`O_RDWR` `O_RDONLY|O_WRONLY`	Richiede un accesso in lettura e scrittura (la combinazione di '**`R_RDONLY`**' e di '**`O_WRONLY`**' è equivalente all'uso di '**`O_RDWR`**').
`O_CREAT`	Richiede di creare contestualmente il file, ma in tal caso va usato anche il parametro ***mode***.
`O_TRUNC`	Se file da aprire esiste già, richiede che questo sia ridotto preventivamente a un file vuoto.
`O_APPEND`	Fa in modo che le operazioni di scrittura avvengano sempre partendo dalla fine del file.

Quando si utilizza l'opzione ***O_CREAT***, è necessario stabilire la modalità dei permessi, attraverso la combinazione di macrovariabili appropriate, come elencato nella tabella successiva. Tale combinazione va fatta con l'uso dell'operatore OR binario; per esempio: '**`S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH`**'. Va osservato che os16 non gestisce i gruppi di utenti, pertanto, la definizione dei permessi relativi agli utenti appartenenti al gruppo proprietario di un file, non ha poi effetti pratici nel controllo degli accessi per tale tipo di contesto.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	00700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	00400_8	Lettura per l'utente proprietario.
`S_IWUSR`	00200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	00100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXG`	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	00040_8	Lettura per il gruppo.
`S_IWGRP`	00020_8	Scrittura per il gruppo.
`S_IXGRP`	00010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_OPEN`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_open()*** è *open(2)* [u0.28].

VALORE RESTITUITO

La funzione restituisce il numero del descrittore del file aperto,

se l'operazione ha avuto successo, altrimenti dà semplicemente `-1`, impostando di conseguenza il valore della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

'`lib/fcntl/open.c`' [i161.4.3]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_open.c`' [i160.4.8]

VEDERE ANCHE

open(2) [u0.28], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *path_full(9)* [i159.3.35], *path_inode_link(9)* [i159.3.37], *inode_truncate(9)* [i159.3.28], *inode_check(9)* [i159.3.16], *file_reference(9)* [i159.3.13], *fd_reference(9)* [i159.3.10].

os16: fd_read(9)

NOME

'**`fd_read`**' - lettura di descrittore di file

SINTASSI

```
<kernel/fs.h>
ssize_t fd_read (pid_t pid, int fdn, void *buffer,
                 size_t count,
                 int *eof);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Il numero del descrittore di file.
`void *`***buffer***	Area di memoria in cui scrivere ciò che viene letto dal descrittore di file.
`size_t` ***count***	Quantità di byte da leggere.
`int *`***eof***	Puntatore a una variabile in cui annotare, eventualmente, il raggiungimento della fine del file.

DESCRIZIONE

La funzione ***fd_read()*** cerca di leggere il file rappresentato dal descrittore ***fdn***, partendo dalla posizione in cui si trova l'indice interno di accesso, per un massimo di ***count*** byte, collocando i dati letti in memoria a partire dal puntatore ***buffer***. L'indice interno al file viene fatto avanzare della quantità di byte letti effettivamente, se invece si incontra la fine del file, viene aggiornata la variabile ****eof***.

La funzione può leggere file normali, file di dispositivo e directory, trattandole però come se fossero dei file puri e semplici. Gli altri tipi di file non sono gestiti da os16.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_READ`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_read()*** è *read(2)* [u0.29].

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti effettivamente, oppure zero se è stata raggiunta la fine del file e non si può proseguire oltre. Va osservato che la lettura effettiva di una quantità inferiore di byte rispetto a quanto richiesto non costituisce un errore: in quel caso i byte mancanti vanno richiesti eventualmente con successive operazioni di lettura. In caso di errore, la funzione restituisce il valore `-1`, aggiornando contestualmente la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in lettura.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os16.

FILE SORGENTI

'`lib/unistd/read.c`' [i161.17.28]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_read.c`' [i160.4.9]

VEDERE ANCHE

read(2) [u0.29], *sysroutine(9)* [i159.8.28], *fd_reference(9)* [i159.3.10], *dev_io(9)* [i159.1.1], *inode_file_read(9)* [i159.3.18].

os16: fd_reference(9)

NOME

'**`fd_reference`**' - riferimento a un elemento della tabella dei descrittori

SINTASSI

```
<kernel/fs.h>
fd_t *fd_reference (pid_t pid, int *fdn);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Il numero del descrittore di file.

DESCRIZIONE

La funzione ***fd_reference()*** restituisce il puntatore all'elemento della tabella dei descrittori, corrispondente al processo e al numero di descrittore specificati. Se però viene fornito un numero di descrittore negativo, si ottiene il puntatore al primo elemento che risulta libero nella tabella.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei descrittori, oppure il puntatore nullo in caso di errore, ma **senza**

aggiornare la variabile ***errno*** del kernel. Infatti, l'unico errore che può verificarsi consiste nel non poter trovare il descrittore richiesto.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_reference.c`' [i160.4.10]

VEDERE ANCHE

file_reference(9) [i159.3.13], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *proc_reference(9)* [i159.8.7].

os16: fd_stat(9)

Vedere *stat(9)* [i159.3.50].

os16: fd_write(9)

NOME

'**`fd_write`**' - scrittura di un descrittore di file

SINTASSI

```
<kernel/fs.h>
ssize_t fd_write (pid_t pid, int fdn, const void *buffer,
                  size_t count);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Il numero del descrittore di file.
`const void *`***buffer***	Area di memoria da cui attingere i dati da scrivere nel descrittore di file.
`size_t` ***count***	Quantità di byte da scrivere.

DESCRIZIONE

La funzione ***fd_write()*** consente di scrivere fino a un massimo di ***count*** byte, tratti dall'area di memoria che inizia all'indirizzo ***buffer***, presso il file rappresentato dal descrittore ***fdn***, del processo ***pid***. La scrittura avviene a partire dalla posizione in cui si trova l'indice interno.

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_WRITE`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_write()*** è *write(2)* [u0.44].

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti effettivamente e in tal caso è possibile anche ottenere una quantità pari a zero. Se si verifica invece un errore, la funzione restituisce `-1` e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in scrittura.
`EISDIR`	Il file è una directory.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os16.
`EIO`	Errore di input-output.

FILE SORGENTI

'`lib/unistd/write.c`' [i161.17.36]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_write.c`' [i160.4.12]

VEDERE ANCHE

write(2) [u0.44], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *dev_io(9)* [i159.1.1], *inode_file_write(9)* [i159.3.19].

os16: file_reference(9)

NOME

'**`file_reference`**' - riferimento a un elemento della tabella dei file di sistema

SINTASSI

```
<kernel/fs.h>
file_t *file_reference (int fno);
```

ARGOMENTI

Argomento	Descrizione
`int` ***fno***	Il numero della voce della tabella dei file, a partire da zero.

DESCRIZIONE

La funzione ***file_reference()*** restituisce il puntatore all'elemento della tabella dei file di sistema, corrispondente al numero indicato come argomento. Se però tale numero fosse negativo, viene restituito il puntatore al primo elemento libero.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei file di sistema, oppure il puntatore nullo in caso di errore, ma **senza aggiornare** la variabile ***errno*** del kernel. Infatti, l'unico errore che può verificarsi consiste nel non poter trovare la voce richiesta.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/file_table.c`' [i160.4.15]

'`kernel/fs/file_reference.c`' [i160.4.13]

VEDERE ANCHE

fd_reference(9) [i159.3.10], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *proc_reference(9)* [i159.8.7].

os16: file_stdio_dev_make(9)

NOME

'**`file_stdio_dev_make`**' - creazione di una voce relativa a un dispositivo di input-output standard, nella tabella dei file di sistema

SINTASSI

```
<kernel/fs.h>
file_t *file_stdio_dev_make (dev_t device, mode_t mode,
                             int oflags);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Il numero del dispositivo da usare per l'input o l'output.
`mode_t` ***mode***	Tipo di file di dispositivo: è praticamente obbligatorio l'uso di '**`S_IFCHR`**'.
`int` ***oflags***	Modalità di accesso: '**`O_RDONLY`**' oppure '**`O_WRONLY`**'.

DESCRIZIONE

La funzione ***file_stdio_dev_make()*** produce una voce nella tabella dei file di sistema, relativa a un dispositivo di input-output, da usare come flusso standard. In altri termini, serve per creare le voci della tabella dei file, relative a standard input, standard output e standard error.

Per ottenere questo risultato occorre coinvolgere anche la funzione *inode_stdio_dev_make(9)* [i159.3.27], la quale si occupa di predisporre un inode, privo però di un collegamento a un file vero e proprio.

Questa funzione viene usata esclusivamente da *proc_sys_exec(9)* [i159.8.20], per attribuire standard input, standard output e standard error, che non fossero già disponibili.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei file di sistema, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Non è possibile allocare un altro file di sistema.

FILE SORGENTI

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/file_stdio_dev_make.c`’ [i160.4.14]

VEDERE ANCHE

proc_sys_exec(9) [i159.8.20], *inode_stdio_dev_make(9)* [i159.3.27], *file_reference(9)* [i159.3.13], *inode_put(9)* [i159.3.24].

os16: inode_alloc(9)

NOME

‘**`inode_alloc`**’ - allocazione di un inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_alloc (dev_t device, mode_t mode, uid_t uid);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Il numero del dispositivo in cui si trova il file system dove allocare l’inode.
`mode_t` ***mode***	Tipo di file e modalità dei permessi da associare all’inode.
`uid_t` ***uid***	Proprietà dell’inode.

DESCRIZIONE

La funzione ***inode_alloc()*** cerca un inode libero nel file system del dispositivo indicato, quindi lo alloca (lo segna come utilizzato) e lo modifica aggiornando il tipo e la modalità dei permessi, oltre al proprietario del file. Se la funzione riesce nel suo intento, restituisce il puntatore all’inode in memoria, il quale rimane così aperto e disponibile per ulteriori elaborazioni.

Questa funzione viene usata esclusivamente da *path_inode_link(9)* [i159.3.37], per la creazione di un nuovo file.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode di sistema, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore fornito per il parametro ***mode*** non è ammissibile.
`ENODEV`	Il dispositivo non corrisponde ad alcuna voce della tabella dei super blocchi; per esempio, il file system cercato potrebbe non essere ancora stato innestato.
`ENOSPC`	Non è possibile allocare l’inode, per mancanza di spazio.
`ENFILE`	Non c’è spazio nella tabella degli inode.

FILE SORGENTI

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/inode_alloc.c`’ [i160.4.16]

VEDERE ANCHE

path_inode_link(9) [i159.3.37], *sb_reference(9)* [i159.3.47], *inode_get(9)* [i159.3.23], *inode_put(9)* [i159.3.24], *inode_truncate(9)* [i159.3.28], *inode_save(9)* [i159.3.26].

os16: inode_check(9)

NOME

‘**`inode_check`**’ - verifica delle caratteristiche di un inode

SINTASSI

```
<kernel/fs.h>
int inode_check (inode_t *inode, mode_t type, int perm,
                 uid_t uid);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ****inode***	Puntatore a un elemento della tabella degli inode.
`mode_t` ***type***	Tipo di file desiderato. Può trattarsi di ‘**`S_IFBLK`**’, ‘**`S_IFCHR`**’, ‘**`S_IFIFO`**’, ‘**`S_IFREG`**’, ‘**`S_IFDIR`**’, ‘**`S_IFLNK`**’, ‘**`S_IFSOCK`**’, come dichiarato nel file ‘`lib/sys/stat.h`’, tuttavia os16 gestisce solo file di dispositivo, file normali e directory.
`int` ***perm***	Permessi richiesti dall’utente ***uid***, rappresentati nei tre bit meno significativi.
`uid_t` ***uid***	Utente nei confronti del quale vanno verificati i permessi di accesso.

DESCRIZIONE

La funzione ***inode_check()*** verifica che l’inode indicato sia di un certo tipo e abbia i permessi di accesso necessari a un certo utente. Tali permessi vanno rappresentati utilizzando solo gli ultimi tre bit (4 = lettura, 2 = scrittura, 1 = esecuzione o attraversamento) e si riferiscono alla richiesta di accesso all’inode, da parte dell’utente ***uid***, tenendo conto del complesso dei permessi che lo riguardano.

Nel parametro ***type*** è ammessa la sovrapposizione di più tipi validi.

Questa funzione viene usata in varie situazioni, internamente al kernel, per verificare il tipo o l’accessibilità di un file.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Le caratteristiche dell’inode sono compatibili con quanto richiesto.
-1	Le caratteristiche dell’inode non sono compatibili, oppure si è verificato un errore. In ogni caso si ottiene l’aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore di ***inode*** corrisponde a un puntatore nullo.
`E_FILE_TYPE`	Il tipo di file dell’inode non corrisponde a quanto richiesto.
`EACCES`	I permessi di accesso non sono compatibili con la richiesta.

FILE SORGENTI

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/inode_check.c`’ [i160.4.17]

os16: inode_dir_empty(9)

«

NOME

'**inode_dir_empty**' - verifica della presenza di contenuti in una directory

SINTASSI

```
<kernel/fs.h>
int inode_dir_empty (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Puntatore a un elemento della tabella degli inode.

DESCRIZIONE

La funzione ***inode_dir_empty()*** verifica che la directory, a cui si riferisce l'inode a cui punta ***inode***, sia vuota.

VALORE RESTITUITO

Valore	Significato del valore restituito
1	*Vero*: si tratta di una directory vuota.
0	*Falso*: se è effettivamente una directory, questa non è vuota, altrimenti non è nemmeno una directory.

ERRORI

Dal momento che un risultato *Falso* non rappresenta necessariamente un errore, per verificare il contenuto della variabile ***errno***, prima dell'uso della funzione occorre azzerarla.

Valore di ***errno***	Significato
`EINVAL`	L'inode non riguarda una directory.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_dir_empty.c`' [i160.4.18]

VEDERE ANCHE

inode_file_read(9) [i159.3.18].

os16: inode_file_read(9)

«

NOME

'**inode_file_read**' - lettura di un file rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_file_read (inode_t *inode, off_t offset,
                         void *buffer, size_t count,
                         int *eof);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Puntatore a un elemento della tabella degli inode, che rappresenta il file da leggere.
`off_t offset`	Posizione, riferita all'inizio del file, a partire dalla quale eseguire la lettura.
`void *buffer`	Puntatore all'area di memoria in cui scrivere ciò che si ottiene dalla lettura del file.
`size_t count`	Quantità massima di byte da leggere.
`int *eof`	Puntatore a un indicatore di fine file, da aggiornare (purché sia un puntatore valido) in base all'esito della lettura.

DESCRIZIONE

La funzione ***inode_file_read()*** legge il contenuto del file a cui si riferisce l'inode ***inode*** e se il puntatore ***eof*** è valido, aggiorna anche la variabile ****eof***.

Questa funzione si avvale a sua volta di *inode_fzones_read(9)* [i159.3.21], per accedere ai contenuti del file, suddivisi in zone, secondo l'organizzazione del file system Minix 1.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti e resi effettivamente disponibili a partire da ciò a cui punta ***buffer***. Se la variabile ***var*** è un puntatore valido, aggiorna anche il suo valore, azzerandolo se la lettura avviene in una posizione interna al file, oppure impostandolo a uno se la lettura richiesta è oltre la fine del file. Se invece si tenta una lettura con un valore di ***offset*** negativo, o specificando il puntatore nullo al posto dell'inode, la funzione restituisce `-1` e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido, oppure il valore di ***offset*** è negativo.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_file_read.c`' [i160.4.19]

VEDERE ANCHE

inode_fzones_read(9) [i159.3.21].

os16: inode_file_write(9)

«

NOME

'**inode_file_write**' - scrittura di un file rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_file_write (inode_t *inode, off_t offset,
                          void *buffer, size_t count);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Puntatore a un elemento della tabella degli inode, che rappresenta il file da scrivere.
`off_t offset`	Posizione, riferita all'inizio del file, a partire dalla quale eseguire la scrittura.
`void *buffer`	Puntatore all'area di memoria da cui trarre i dati da scrivere nel file.
`size_t count`	Quantità massima di byte da scrivere.

DESCRIZIONE

La funzione ***inode_file_write()*** scrive nel file rappresentato da ***inode***, a partire dalla posizione ***offset*** (purché non sia un valore negativo), la quantità massima di byte indicati con ***count***, ciò che si trova in memoria a partire da ***buffer***.

Questa funzione si avvale a sua volta di *inode_fzones_read(9)* [i159.3.21], per accedere ai contenuti del file, suddivisi in zone, secondo l'organizzazione del file system Minix 1, e di *zone_write(9)* [i159.3.53], per la riscrittura delle zone relative.

Per os16, le operazioni di scrittura nel file system sono sincrone, senza alcun trattenimento in memoria (ovvero senza *cache*).

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti. La scrittura può avvenire oltre la fine del file, anche in modo discontinuo; tuttavia, non è ammissibile un valore di ***offset*** negativo.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido, oppure il valore di ***offset*** è negativo.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_file_write.c`' [i160.4.20]

VEDERE ANCHE

inode_fzones_read(9) [i159.3.21], *zone_write(9)* [i159.3.53].

os16: inode_free(9)

NOME

'**`inode_free`**' - deallocazione di un inode

SINTASSI

```
<kernel/fs.h>
int inode_free (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ***inode***	Puntatore a un elemento della tabella degli inode.

DESCRIZIONE

La funzione ***inode_free()*** libera l'inode specificato attraverso il puntatore ***inode***, rispetto al proprio super blocco. L'operazione comporta semplicemente il fatto di indicare questo inode come libero, senza controlli per verificare se effettivamente non esistono più collegamenti nel file system che lo riguardano.

Questa funzione viene usata esclusivamente da *inode_put(9)* [i159.3.24], per completare la cancellazione di un inode che non ha più collegamenti nel file system, nel momento in cui non vi si fa più riferimento nel sistema in funzione.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_free.c`' [i160.4.21]

VEDERE ANCHE

inode_save(9) [i159.3.26], *inode_alloc(9)* [i159.3.15].

os16: inode_fzones_read(9)

NOME

'**`inode_fzones_read`**', '**`inode_fzones_write`**' - lettura e scrittura di zone relative al contenuto di un file

SINTASSI

```
<kernel/fs.h>
blkcnt_t inode_fzones_read  (inode_t *inode, zno_t zone_start,
                             void *buffer, blkcnt_t blkcnt);
blkcnt_t inode_fzones_write (inode_t *inode, zno_t zone_start,
                             void *buffer, blkcnt_t blkcnt);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ***inode***	Puntatore a un elemento della tabella degli inode, con cui si individua il file da cui leggere o in cui scrivere.
`zno_t` ***zone_start***	Il numero di zona, relativo al file, a partire dalla quale iniziare la lettura o la scrittura.
`void *`***buffer***	Il puntatore a un'area di memoria tampone, da usare per depositare i dati letti o per trarre i dati da scrivere.
`blkcnt_t` ***blkcnt***	La quantità di zone da leggere o scrivere.

DESCRIZIONE

Le funzioni ***inode_fzones_read()*** e ***inode_fzones_write()***, consentono di leggere e di scrivere un file, a zone intere (la zona è un multiplo del blocco, secondo la filosofia del file system Minix 1).

Questa funzione vengono usate soltanto da *inode_file_read(9)* [i159.3.18] e *inode_file_write(9)* [i159.3.19], con le quali l'accesso ai file si semplifica a livello di byte.

VALORE RESTITUITO

Le due funzioni restituiscono la quantità di zone lette o scritte effettivamente. Una quantità pari a zero potrebbe eventualmente rappresentare un errore, ma solo in alcuni casi. Per poterlo verificare, occorre azzerare la variabile ***errno*** prima di chiamare le funzioni, riservandosi di verificarne successivamente il valore.

ERRORI

Valore di ***errno***	Significato
`EIO`	L'accesso alla zona richiesta non è potuto avvenire.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_fzones_read.c`' [i160.4.22]

'`kernel/fs/inode_fzones_write.c`' [i160.4.23]

VEDERE ANCHE

inode_file_read(9) [i159.3.18], *inode_file_write(9)* [i159.3.19], *zone_read(9)* [i159.3.53], *zone_write(9)* [i159.3.53].

os16: inode_fzones_write(9)

Vedere *inode_fzones_read(9)* [i159.3.21].

os16: inode_get(9)

NOME

'**`inode_get`**' - caricamento di un inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_get (dev_t device, ino_t ino);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo riferito a un'unità di memorizzazione, dove cercare l'inode numero ***ino***.
`ino_t` ***ino***	Numero di inode, relativo al file system contenuto nell'unità ***device***.

DESCRIZIONE

La funzione ***inode_get()*** consente di «aprire» un inode, fornendo il numero del dispositivo corrispondente all'unità di memorizzazione e il numero dell'inode del file system in essa contenuto. L'inode in questione potrebbe essere già stato aperto e quindi già disponibile in memoria nella tabella degli inode; in tal caso, la funzione si limita a incrementare il contatore dei riferimenti a tale inode, da parte del sistema in funzione, restituendo il puntatore all'elemento della tabella che lo contiene gia. Se invece l'inode

non è ancora presente nella tabella rispettiva, la funzione deve provvedere a caricarlo.

Se si richiede un inode non ancora disponibile, contenuto in un'unità di cui non è ancora stato caricato il super blocco nella tabella rispettiva, la funzione deve provvedere anche a questo procedimento.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che rappresenta l'inode aperto. Se però si presenta un problema, restituisce il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EUNKNOWN`	Si tratta di un problema imprevisto e non meglio identificabile.
`ENFILE`	La tabella degli inode è già occupata completamente e non è possibile aprirne altri.
`ENODEV`	Il dispositivo richiesto non è valido.
`ENOENT`	Il numero di inode richiesto non esiste.
`EIO`	Errore nella lettura del file system.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_get.c`' [i160.4.24]

VEDERE ANCHE

offsetof (3) [u0.75], *inode_put(9)* [i159.3.24], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *sb_inode_status(9)* [i159.3.45], *dev_io(9)* [i159.1.1].

os16: inode_put(9)

NOME

'**`inode_put`**' - rilascio di un inode

SINTASSI

```
<kernel/fs.h>
int inode_put (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ***inode***	Puntatore a un elemento della tabella di inode.

DESCRIZIONE

La funzione ***inode_put()*** «chiude» un inode, riducendo il contatore degli accessi allo stesso. Tuttavia, se questo contatore, dopo il decremento, raggiunge lo zero, è necessario verificare se nel frattempo anche i collegamenti del file system si sono azzerati, perché in tal caso occorre anche rimuovere l'inode, nel senso di segnalarlo come libero per la creazione di un nuovo file. In ogni caso, le informazioni aggiornate dell'inode, ancora allocato o liberato, vengono memorizzate nel file system.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il puntatore ***inode*** non è valido.
`EUNKNOWN`	Si tratta di un problema imprevisto e non meglio identificabile.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_put.c`' [i160.4.25]

VEDERE ANCHE

inode_truncate(9) [i159.3.28], *inode_free(9)* [i159.3.20], *inode_save(9)* [i159.3.26].

os16: inode_reference(9)

NOME

'**`inode_reference`**' - riferimento a un elemento della tabella di inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_reference (dev_t device, ino_t ino);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo riferito a un'unità di memorizzazione, dove cercare l'inode numero ***ino***.
`ino_t` ***ino***	Numero di inode, relativo al file system contenuto nell'unità ***device***.

DESCRIZIONE

La funzione ***inode_reference()*** cerca nella tabella degli inode la voce corrispondente ai dati forniti come argomenti, ovvero quella dell'inode numero ***ino*** del file system contenuto nel dispositivo ***device***, restituendo il puntatore alla voce corrispondente. Tuttavia ci sono dei casi particolari:

- se il numero del dispositivo e quello dell'inode sono entrambi zero, viene restituito il puntatore all'inizio della tabella, ovvero al primo elemento della stessa;
- se il numero del dispositivo e quello dell'inode sono pari a un numero negativo (rispettivamente '**`(dev_t) -1`**' e '**`(ino_t) -1`**'), viene restituito il puntatore alla prima voce libera;
- se il numero del dispositivo è pari a zero e il numero dell'inode è pari a uno, si intende ricercare la voce dell'inode della directory radice del file system principale.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode, se la ricerca si compie con successo. In caso di problemi, invece, la funzione restituisce il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`E_CANNOT_FIND_ROOT_DEVICE`	Nella tabella dei super blocchi non è possibile trovare il file system principale.
`E_CANNOT_FIND_ROOT_INODE`	Nella tabella degli inode non è possibile trovare la directory radice del file system principale.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_reference.c`' [i160.4.26]

VEDERE ANCHE

sb_reference(9) [i159.3.47], *file_reference(9)* [i159.3.13], *proc_reference(9)* [i159.8.7].

os16: inode_save(9)

NOME

'**`inode_save`**' - memorizzazione dei dati di un inode

SINTASSI

```
<kernel/fs.h>
int inode_save (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a una voce della tabella degli inode.

DESCRIZIONE

La funzione ***inode_save()*** memorizza l'inode a cui si riferisce la voce ****inode***, nel file system, ammesso che si tratti effettivamente di un inode relativo a un file system e che sia stato modificato dopo l'ultima memorizzazione precedente. In questo caso, la funzione, a sua volta, richiede la memorizzazione del super blocco.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il puntatore ***inode*** è nullo.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_save.c`' [i160.4.27]

VEDERE ANCHE

sb_save(9) [i159.3.48], *dev_io(9)* [i159.1.1].

os16: inode_stdio_dev_make(9)

«

NOME

'**`inode_stdio_dev_make`**' - creazione di una voce relativa a un dispositivo di input-output standard, nella tabella degli inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_stdio_dev_make (dev_t device, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Il numero del dispositivo da usare per l'input o l'output.
`mode_t` ***mode***	Tipo di file di dispositivo: è praticamente obbligatorio l'uso di '**`S_IFCHR`**'.

DESCRIZIONE

La funzione ***inode_stdio_dev_make()*** produce una voce nella tabella degli inode, relativa a un dispositivo di input-output, da usare come flusso standard. In altri termini, serve per creare le voci della tabella degli inode, relative a standard input, standard output e standard error.

Questa funzione viene usata esclusivamente da *file_stdio_dev_make(9)* [i159.3.14], per creare una voce da usare come flusso standard di input o di output, nella tabella dei file.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti della chiamata non sono validi.
`ENFILE`	Non è possibile allocare un altro inode.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_stdio_dev_make.c`' [i160.4.28]

VEDERE ANCHE

file_stdio_dev_make(9) [i159.3.14], *inode_reference(9)* [i159.3.25].

os16: inode_truncate(9)

«

NOME

'**`inode_truncate`**' - troncamento del file a cui si riferisce un inode

SINTASSI

```
<kernel/fs.h>
int inode_truncate (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a una voce della tabella di inode.

DESCRIZIONE

La funzione ***inode_truncate()*** richiede che il puntatore ***inode*** si riferisca a una voce della tabella degli inode, relativa a un file contenuto in un file system. Lo scopo della funzione è annullare il contenuto di tale file, trasformandolo in un file vuoto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Allo stato attuale dello sviluppo della funzione, non ci sono controlli e non sono previsti errori.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/inode_truncate.c`' [i160.4.30]

VEDERE ANCHE

zone_free(9) [i159.3.51], *sb_save(9)* [i159.3.48], *inode_save(9)* [i159.3.26].

os16: inode_zone(9)

«

NOME

'**`inode_zone`**' - traduzione del numero di zona relativo in un numero di zona assoluto

SINTASSI

```
<kernel/fs.h>
zno_t inode_zone (inode_t *inode, zno_t fzone, int write);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a una voce della tabella di inode.
`zno_t` ***fzone***	Numero di zona relativo al file dell'inode preso in considerazione.
`int` ***write***	Valore da intendersi come *Vero* o *Falso*, con cui consentire o meno la creazione al volo di una zona mancante.

DESCRIZIONE

La funzione ***inode_zone()*** serve a tradurre il numero di una zona, inteso relativamente a un file, nel numero assoluto relativamente al file system in cui si trova. Tuttavia, un file può essere memorizzato effettivamente in modo discontinuo, ovvero con zone inesistenti nella sua parte centrale. Il contenuto di un file che non dispone effettivamente di zone allocate, corrisponde a un contenuto nullo dal punto di vista binario (zero binario), ma per la funzione, una zona assente comporta la restituzione di un valore nullo, perché nel file system non c'è. Pertanto, se l'argomento

corrispondente al parametro ***write*** contiene un valore diverso da zero, la funzione che non trova una zona, la alloca e quindi ne restituisce il numero.

VALORE RESTITUITO

La funzione restituisce il numero della zona che nel file system corrisponde a quella relativa richiesta per un certo file. Nel caso la zona non esista, perché non allocata, restituisce zero. Tuttavia, la zona zero di un file system Minix 1 esiste, ma contiene sostanzialmente le informazioni amministrative del super blocco, pertanto non può essere una traduzione valida di una zona di un file.

ERRORI

La funzione non prevede il verificarsi di errori.

FILE SORGENTI

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/inode_zone.c`’ [i160.4.31]

VEDERE ANCHE

memset(3) [u0.72], *zone_alloc(9)* [i159.3.51], *zone_read(9)* [i159.3.53], *zone_write(9)* [i159.3.53].

os16: path_chdir(9)

NOME

‘**`path_chdir`**’ - cambiamento della directory corrente

SINTASSI

```
<kernel/fs.h>
int path_chdir (pid_t pid, const char *path);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso della nuova directory corrente, riferito alla directory corrente del processo ***pid***.

DESCRIZIONE

La funzione ***path_chdir()*** cambia la directory corrente del processo ***pid***, in modo che quella nuova corrisponda al percorso annotato nella stringa ***path***.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_CHDIR`**’. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chdir()*** è *chdir(2)* [u0.3].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`EACCES`	Accesso negato.
`ENOTDIR`	Uno dei componenti del percorso non è una directory.
`ENOENT`	Uno dei componenti del percorso non esiste.

FILE SORGENTI

‘`lib/unistd/chdir.c`’ [i161.17.3]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/path_chdir.c`’ [i160.4.32]

VEDERE ANCHE

chdir(2) [u0.3], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_full(9)* [i159.3.35], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: path_chmod(9)

NOME

‘**`path_chmod`**’ - cambiamento della modalità dei permessi di un file

SINTASSI

```
<kernel/fs.h>
int path_chmod (pid_t pid, const char *path, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file su cui intervenire.
`mode_t` ***mode***	La modalità dei_permessi da applicare (contano solo i 12 bit meno significativi).

DESCRIZIONE

La funzione ***path_chmod()*** modifica la modalità dei permessi di accesso del file indicato, tramite il suo percorso, relativo eventualmente alla directory corrente del processo ***pid***.

Tradizionalmente, i permessi si scrivono attraverso un numero in base otto; in alternativa, si possono usare convenientemente della macro-variabili, dichiarate nel file ‘`lib/sys/stat.h`’, combinate assieme con l’operatore binario OR.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	00700_8	Lettura, scrittura ed esecuzione per l’utente proprietario.
`S_IRUSR`	00400_8	Lettura per l’utente proprietario.
`S_IWUSR`	00200_8	Scrittura per l’utente proprietario.
`S_IXUSR`	00100_8	Esecuzione per l’utente proprietario.
`S_IRWXG`	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	00040_8	Lettura per il gruppo.
`S_IWGRP`	00020_8	Scrittura per il gruppo.
`S_IXGRP`	00010_8	Esecuzione per il gruppo.
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky, che nella tabella non sono stati nemmeno annotati; inoltre, non tiene in considerazione i permessi legati al gruppo, perché non tiene traccia dei gruppi.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_CHMOD**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chmod()*** è *chmod(2)* [u0.4].

FILE SORGENTI

'lib/sys/stat/chmod.c' [i161.13.1]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/fs.h' [u0.4]

'kernel/fs/path_chmod.c' [i160.4.33]

VEDERE ANCHE

chmod(2) [u0.4], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36].

os16: path_chown(9)

NOME

'**path_chown**' - cambiamento della proprietà di un file

SINTASSI

```
<kernel/fs.h>
int path_chown (pid_t pid, const char *path, uid_t uid,
                gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
const char ***path**	Il percorso del file su cui intervenire.
uid_t ***uid***	Utente a cui attribuire la proprietà del file.
gid_t ***gid***	Gruppo a cui associare il file.

DESCRIZIONE

La funzione ***path_chown()*** modifica la proprietà di un file, fornendo il numero UID e il numero GID. Il file viene indicato attraverso il percorso scritto in una stringa, relativo alla directory corrente del processo ***pid***.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_CHOWN**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chown()*** è *chown(2)* [u0.5].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EINVAL	Argomento non valido.
EPERM	Permessi insufficienti per eseguire l'operazione.
ENOTDIR	Uno dei componenti del percorso non è una directory.
ENOENT	Uno dei componenti del percorso non esiste.
EBADF	Il descrittore del file non è valido.

DIFETTI

Benché sia consentito di attribuire il numero del gruppo, os16 non valuta i permessi di accesso ai file, relativi a questi.

FILE SORGENTI

'lib/unistd/chown.c' [i161.17.4]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/fs.h' [u0.4]

'kernel/fs/path_chown.c' [i160.4.34]

VEDERE ANCHE

chown(2) [u0.5], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_save(9)* [i159.3.26], *inode_put(9)* [i159.3.24].

os16: path_device(9)

NOME

'**path_device**' - conversione di un file di dispositivo nel numero corrispondente

SINTASSI

```
<kernel/fs.h>
dev_t path_device (pid_t pid, const char *path);
```

ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Processo elaborativo per conto del quale si agisce.
const char ***path**	Il percorso del file di dispositivo.

DESCRIZIONE

La funzione ***path_device()*** consente di trarre il numero complessivo di un dispositivo, a partire da un file di dispositivo.

Questa funzione viene usata soltanto da *path_mount(9)* [i159.8.28].

VALORE RESTITUITO

La funzione restituisce il numero del dispositivo corrispondente al file indicato, oppure il valore -1, in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
ENODEV	Il file richiesto non è un file di dispositivo.
ENOENT	Il file richiesto non esiste.
EACCES	Il file richiesto non è accessibile secondo i privilegi del processo ***pid***.

FILE SORGENTI

'kernel/fs.h' [u0.4]

'kernel/fs/path_device.c' [i160.4.35]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: path_fix(9)

NOME

'**path_fix**' - semplificazione di un percorso

SINTASSI

```
<kernel/fs.h>
int path_fix (char *path);
```

ARGOMENTI

Argomento	Descrizione
char ***path**	Il percorso da semplificare.

DESCRIZIONE

La funzione ***path_fix()*** legge la stringa del percorso ***path*** e la rielabora, semplificandolo. La semplificazione riguarda l'eliminazione di riferimenti inutili alla directory corrente e di indietreggiamenti. Il percorso può essere assoluto o relativo: la funzione non ne cambia l'origine.

VALORE RESTITUITO

La funzione restituisce sempre zero e non è prevista la manifestazione di errori.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_fix.c`' [i160.4.36]

VEDERE ANCHE

strtok(3) [u0.120], *strcmp(3)* [u0.106], *strcat(3)* [u0.104], *strncat(3)* [u0.104], *strncpy(3)* [u0.108].

os16: path_full(9)

«

NOME

'**`path_full`**' - traduzione di un percorso relativo in un percorso assoluto

SINTASSI

```
<kernel/fs.h>
int path_full (const char *path, const char *path_cwd,
               char *full_path);
```

ARGOMENTI

Argomento	Descrizione
`const char *`***path***	Il percorso relativo alla posizione ***path_cwd***.
`const char *`***path_cwd***	La directory corrente.
`char *`***full_path***	Il luogo in cui scrivere il percorso assoluto.

DESCRIZIONE

La funzione ***path_full()*** ricostruisce un percorso assoluto, mettendolo in memoria a partire da ciò a cui punta ***full_path***.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'insieme degli argomenti non è valido.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_full.c`' [i160.4.37]

VEDERE ANCHE

strtok(3) [u0.120], *strcmp(3)* [u0.106], *strcat(3)* [u0.104], *strncat(3)* [u0.104], *strncpy(3)* [u0.108], *path_fix(9)* [i159.3.34].

os16: path_inode(9)

«

NOME

'**`path_inode`**' - caricamento di un inode, partendo dal percorso del file

SINTASSI

```
<kernel/fs.h>
inode_t *path_inode (pid_t pid, const char *path);
```


ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file del quale si vuole ottenere l'inode.

DESCRIZIONE

La funzione ***path_inode()*** carica un inode nella tabella degli inode, oppure lo localizza se questo è già caricato, partendo dal percorso di un file. L'operazione è subordinata all'accessibilità del percorso che conduce al file, nel senso che il processo ***pid*** deve avere il permesso di accesso («x») in tutti gli stadi dello stesso.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che contiene le informazioni caricate in memoria sull'inode. Se qualcosa non va, restituisce invece il puntatore nullo, aggiornando di conseguenza il contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENOENT`	Uno dei componenti del percorso non esiste.
`ENFILE`	Non è possibile allocare un inode ulteriore, perché la tabella è già occupata completamente.
`EIO`	Error di input-output.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_inode.c`' [i160.4.38]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_full(9)* [i159.3.35], *inode_get(9)* [i159.3.23], *inode_put(9)* [i159.3.24], *inode_check(9)* [i159.3.16], *inode_file_read(9)* [i159.3.18].

os16: path_inode_link(9)

«

NOME

'**`path_inode_link`**' - creazione di un collegamento fisico o di un nuovo file

SINTASSI

```
<kernel/fs.h>
inode_t *path_inode_link (pid_t pid, const char *path,
                          inode_t *inode, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file per il quale si vuole creare il collegamento fisico.
`inode_t *`***inode***	Puntatore a una voce della tabella degli inode, alla quale si vuole collegare il nuovo file.
`mode_t` ***mode***	Nel caso l'inode non sia stato fornito, dovendo creare un nuovo file, questo parametro richiede il tipo e i permessi del file da creare.

DESCRIZIONE

La funzione ***path_inode_link()*** crea un collegamento fisico con il nome fornito in ***path***, riferito all'inode a cui punta ***inode***. Tuttavia, l'argomento corrispondente al parametro ***inode*** può essere un puntatore nullo, e in tal caso viene creato un file vuoto, allocando contestualmente un nuovo inode, usando l'argomento corrispondente al parametro ***mode*** per il tipo e la modalità dei permessi del nuovo file.

Il processo ***pid*** deve avere i permessi di accesso per tutte le directory che portano al file da collegare o da creare; inoltre, nell'ultima directory ci deve essere anche il permesso di scrittura, dovendo intervenire sulla stessa modificandola.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che descrive l'inode collegato o creato. In caso di problemi, restituisce invece il puntatore nullo, aggiornando di conseguenza il contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'insieme degli argomenti non è valido: se l'inode è stato indicato, il parametro ***mode*** deve essere nullo; al contrario, se l'inode non è specificato, il parametro ***mode*** deve contenere informazioni valide.
`EPERM`	Non è possibile creare il collegamento di un inode corrispondente a una directory.
`EMLINK`	Non è possibile creare altri collegamenti all'inode, il quale ha già raggiunto la quantità massima.
`EEXIST`	Il file ***path*** esiste già.
`EACCES`	Impossibile accedere al percorso che dovrebbe contenere il file da collegare.
`EROFS`	Il file system è innestato in sola lettura e non si può creare il collegamento.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_inode_link.c`' [i160.4.39]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_inode(9)* [i159.3.36], *inode_get(9)* [i159.3.23], *inode_put(9)* [i159.3.24], *inode_save(9)* [i159.3.26], *inode_check(9)* [i159.3.16], *inode_alloc(9)* [i159.3.15], *inode_file_read(9)* [i159.3.18], *inode_file_write(9)* [i159.3.19].

NOME

'**`path_link`**' - creazione di un collegamento fisico

SINTASSI

```
<kernel/fs.h>
int path_link (pid_t pid, const char *path_old,
               const char *path_new);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path_old***	Il percorso del file originario.
`const char *`***path_new***	Il percorso del collegamento da creare.

DESCRIZIONE

La funzione ***path_link()*** produce un nuovo collegamento a un file già esistente. Va fornito il percorso del file già esistente, ***path_old*** e quello del file da creare, in qualità di collegamento, ***path_new***. L'operazione può avvenire soltanto se i due percorsi si trovano sulla stessa unità di memorizzazione e se ci sono i permessi di scrittura necessari nella directory di destinazione per il processo ***pid***. Dopo l'operazione di collegamento, fatta in questo modo, non è possibile distinguere quale sia stato il file originale e quale sia invece il nome aggiunto.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_LINK`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_link()*** è *link(2)* [u0.23].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il nome da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Il file system consente soltanto un accesso in lettura.
`ENOTDIR`	Uno dei due percorsi non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.

FILE SORGENTI

'`lib/unistd/link.c`' [i161.17.26]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs/path_link.c`' [i160.4.40]

'`kernel/fs.h`' [u0.4]

VEDERE ANCHE

link(2) [u0.23], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *path_inode_link(9)* [i159.3.37], *inode_put(9)* [i159.3.24].

NOME

'**`path_mkdir`**' - creazione di una directory

SINTASSI

```
<kernel/fs.h>
int path_mkdir (pid_t pid, const char *path, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso della directory da creare.
`mode_t` ***mode***	Modalità dei permessi da attribuire alla nuova directory.

DESCRIZIONE

La funzione ***path_mkdir()*** crea una directory, indicata attraverso un percorso (parametro ***path()***) e specificando la modalità dei permessi (parametro ***mode***). Va osservato che il valore del parametro ***mode*** non viene preso in considerazione integralmente: di questo si considerano solo gli ultimi nove bit, ovvero quelli dei permessi di utenti, gruppi e altri utenti; inoltre, vengono tolti i bit presenti nella maschera dei permessi associata al processo.

La directory che viene creata in questo modo, appartiene all'identità efficace del processo, ovvero all'utente per conto del quale questo sta funzionando.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_MKDIR`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_mkdir()*** è *mkdir(2)* [u0.25].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso della directory da creare, non è una directory.
`ENOENT`	Una porzione del percorso della directory da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat/mkdir.c`’ [i161.13.4]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_mkdir.c`’ [i160.4.41]

VEDERE ANCHE

mkdir(2) [u0.25], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_file_write(9)* [i159.3.19], *inode_put(9)* [i159.3.24].

os16: path_mknod(9)

NOME

‘**`path_mknod`**’ - creazione di un file vuoto di qualunque tipo

SINTASSI

```
<kernel/fs.h>
int path_mknod (pid_t pid, const char *path, mode_t mode,
                dev_t device);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file da creare.
`mode_t` ***mode***	Tipo e modalità dei permessi del nuovo file.
`dev_t` ***device***	Numero di dispositivo nel caso il tipo sia riferito a un file di dispositivo.

DESCRIZIONE

La funzione ***path_mknod()*** crea un file vuoto, di qualunque tipo. Potenzialmente può creare anche una directory, ma priva di qualunque voce, rendendola così non adeguata al suo scopo (una directory richiede almeno le voci ‘`.`’ e ‘`..`’, per potersi considerare tale).

Il parametro ***path*** specifica il percorso del file da creare; il parametro ***mode*** serve a indicare il tipo di file da creare, oltre ai permessi comuni.

Il parametro ***device***, con il quale va indicato il numero di un dispositivo (completo di numero primario e secondario), viene preso in considerazione soltanto se nel parametro ***mode*** si richiede la creazione di un file di dispositivo a caratteri o a blocchi.

Il valore del parametro ***mode*** va costruito combinando assieme delle macro-variabili definite nel file ‘`lib/sys/stat.h`’, come descritto nella pagina di manuale *stat(2)* [u0.36], tenendo conto che os16 non può gestire file FIFO, collegamenti simbolici e socket di dominio Unix.

Il valore del parametro ***mode***, per la porzione che riguarda i permessi di accesso al file, viene comunque filtrato con la maschera dei permessi (*umask(2)* [u0.36]).

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_MKNOD`**’. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_mknod()*** è *mknod(2)* [u0.26].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso del file da creare, non è una directory.
`ENOENT`	Una porzione del percorso del file da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]

‘`lib/sys/stat/mknod.c`’ [i161.13.5]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_mknod.c`’ [i160.4.42]

VEDERE ANCHE

mknod(2) [u0.26], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: path_mount(9)

NOME

‘**`path_mount`**’, ‘**`path_umount`**’ - innesto e distacco di un file system

SINTASSI

```
<kernel/fs.h>
int path_mount  (pid_t pid, const char *path_dev,
                 const char *path_mnt, int options);
int path_umount (pid_t pid, const char *path_mnt);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path_dev***	Il file di dispositivo dell’unità da innestare.
`const char *`***path_mnt***	Il percorso della directory di innesto.
`int` ***options***	Opzioni di innesto.

DESCRIZIONE

La funzione ***path_mount()*** permette l’innesto di un’unità di memorizzazione individuata attraverso il percorso del file di dispositivo nel parametro ***path_dev***, nella directory corrispondente al percorso ***path_mnt***, con le opzioni indicate numericamente nell’ultimo argomento ***options***. Le opzioni di innesto, rappresentate attraverso delle macro-variabili, sono solo due:

Opzione	Descrizione
`MOUNT_DEFAULT`	Innesto normale, in lettura e scrittura.
`MOUNT_RO`	Innesto in sola lettura.

La funzione ***path_umount()*** consente di staccare un innesto fatto precedentemente, specificando il percorso della directory in cui questo è avvenuto.

Queste funzioni vengono usate soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento delle chiamate di sistema di tipo '**`SYS_MOUNT`**' e '**`SYS_UMOUNT`**'. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***path_mount()*** e ***path_umount()***, sono *mount(2)* [u0.27] e *umount(2)* [u0.27].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: va verificato il contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Problema di accesso dovuto alla mancanza dei permessi necessari.
`ENOTDIR`	Ciò che dovrebbe essere una directory, non lo è.
`EBUSY`	La directory innesta già un file system e non può innestarne un altro.
`ENOENT`	La directory non esiste.
`E_NOT_MOUNTED`	La directory non innesta un file system (da staccare).
`EUNKNOWN`	Si è verificato un problema, non meglio precisato e non previsto.

FILE SORGENTI

'`lib/sys/os16/mount.c`' [i161.12.12]
'`lib/sys/os16/umount.c`' [i161.12.16]
'`lib/sys/os16/sys.s`' [i161.12.15]
'`kernel/proc/_isr.s`' [i160.9.1]
'`kernel/proc/sysroutine.c`' [i160.9.30]
'`kernel/fs.h`' [u0.4]
'`kernel/fs/path_mount.c`' [i160.4.43]
'`kernel/fs/path_umount.c`' [i160.4.45]

VEDERE ANCHE

mount(2) [u0.27], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_device(9)* [i159.3.33], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24], *sb_mount(9)* [i159.3.46].

os16: path_stat(9)

Vedere *stat(9)* [i159.3.50].

os16: path_umount(9)

Vedere *path_mount(9)* [i159.3.41].

os16: path_unlink(9)

NOME

'**`path_unlink`**' - cancellazione di un nome

SINTASSI

```
<kernel/fs.h>
int path_unlink (pid_t pid, const char *path);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso che rappresenta il file da cancellare.

DESCRIZIONE

La funzione ***path_unlink()*** cancella un nome da una directory, ma se si tratta dell'ultimo collegamento che ha quel file, allora libera anche l'inode corrispondente.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_UNLINK`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_unlink()*** è *unlink(2)* [u0.42].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`ENOTEMPTY`	È stata tentata la cancellazione di una directory, ma questa non è vuota.
`ENOTDIR`	Una delle directory del percorso, non è una directory.
`ENOENT`	Il nome richiesto non esiste.
`EROFS`	Il file system è in sola lettura.
`EPERM`	Mancano i permessi necessari.
`EUNKNOWN`	Si è verificato un errore imprevisto e sconosciuto.

FILE SORGENTI

'`lib/unistd/unlink.c`' [i161.17.35]
'`lib/sys/os16/sys.s`' [i161.12.15]
'`kernel/proc/_isr.s`' [i160.9.1]
'`kernel/proc/sysroutine.c`' [i160.9.30]
'`kernel/fs.h`' [u0.4]
'`kernel/fs/path_unlink.c`' [i160.4.46]

VEDERE ANCHE

unlink(2) [u0.42], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_check(9)* [i159.3.16], *inode_file_read(9)* [i159.3.18], *inode_file_write(9)* [i159.3.19], *inode_put(9)* [i159.3.24].

os16: sb_inode_status(9)

NOME

'**`sb_inode_status`**', '**`sb_zone_status`**' - verifica di utilizzazione attraverso il controllo delle mappe di inode e di zone

SINTASSI

```
<kernel/fs.h>
int sb_inode_status (sb_t *sb, ino_t ino);
int sb_zone_status  (sb_t *sb, zno_t zone);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *`***sb***	Puntatore a una voce della tabella dei super blocchi.
`ino_t` ***ino***	Numero di inode.
`zno_t` ***ino***	Numero di zona.

DESCRIZIONE

La funzione ***sb_inode_status()*** verifica che un certo inode, individuato per numero, risulti utilizzato nel file system a cui si riferisce il super blocco a cui punta il primo argomento.

La funzione ***sb_zone_status()*** verifica che una certa zona, individuato per numero, risulti utilizzata nel file system a cui si riferisce il super blocco a cui punta il primo argomento.

La funzione ***sb_inode_status()*** viene usata soltanto da *inode_get(9)* [i159.3.23]; la funzione ***sb_zone_status()*** non viene usata affatto.

VALORE RESTITUITO

Valore	Significato
1	L'inode o la zona risultano utilizzati.
0	L'inode o la zona risultano liberi (allocabili).
`-1`	Errore: è stato richiesto un numero di inode o di zona pari a zero, oppure ***sb*** è un puntatore nullo.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	È stato richiesto un numero di inode o di zona pari a zero, oppure ***sb*** è un puntatore nullo.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/sb_inode_status.c`' [i160.4.47]

'`kernel/fs/sb_zone_status.c`' [i160.4.52]

VEDERE ANCHE

inode_alloc(9) [i159.3.15], *zone_alloc(9)* [i159.3.51].

os16: sb_mount(9)

NOME

'**`sb_mount`**' - innesto di un dispositivo di memorizzazione

SINTASSI

```
<kernel/fs.h>
sb_t *sb_mount (dev_t device, inode_t **inode_mnt,
                int options);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo da innestare.
`inode_t **`***inode_mnt***	Puntatore di puntatore a una voce della tabella di inode. Il valore di ****inode_mnt*** potrebbe essere un puntatore nullo.
`int` ***options***	Opzioni per l'innesto.

DESCRIZIONE

La funzione ***sb_mount()*** innesta il dispositivo rappresentato numericamente dal primo parametro, sulla directory corrispondente all'inode a cui punta, indirettamente, il secondo parametro, con le opzioni del terzo parametro.

Il secondo parametro è un puntatore di puntatore al tipo '**`inode_t`**', in quanto il valore rappresentato da ****inode_mnt*** deve poter essere modificato dalla funzione. Infatti, quando si vuole innestare il file system principale, si crea una situazione particolare, perché la directory di innesto è la radice dello stesso file system da innestare; pertanto, ****inode_mnt*** deve essere un puntatore nullo ed è compito della funzione far sì che diventi il puntatore alla voce corretta nella tabella degli inode.

Questa funzione viene usata da *proc_init(9)* [i159.8.6] per innestare il file system principale, e da *path_mount(9)* [i159.3.41] per innestare un file system in condizioni diverse.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella dei super blocchi che rappresenta il dispositivo innestato. In caso si insuccesso, restituisce invece il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBUSY`	Il dispositivo richiesto risulta già innestato; la directory di innesto è già utilizzata; la tabella dei super blocchi è già occupata del tutto.
`EIO`	Errore di input-output.
`ENODEV`	Il file system del dispositivo richiesto non può essere gestito.
`E_MAP_INODE_TOO_BIG`	La mappa che rappresenta lo stato di utilizzo degli inode del file system, è troppo grande e non può essere caricata in memoria.
`E_MAP_ZONE_TOO_BIG`	La mappa che rappresenta lo stato di utilizzo delle zone (i blocchi di dati del file system Minix 1) è troppo grande e non può essere caricata in memoria.
`E_DATA_ZONE_TOO_BIG`	Nel file system che si vorrebbe innestare, la dimensione della zona di dati è troppo grande rispetto alle possibilità di os16.
`EUNKNOWN`	Errore imprevisto e sconosciuto.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/sb_mount.c`' [i160.4.48]

VEDERE ANCHE

sb_reference(9) [i159.3.47], *dev_io(9)* [i159.1.1], *inode_get(9)* [i159.3.23].

os16: sb_reference(9)

NOME

'**`sb_reference`**' - riferimento a un elemento della tabella dei super blocchi

SINTASSI

```
<kernel/fs.h>
sb_t *sb_reference (dev_t device);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo di un'unità di memorizzazione di massa.

DESCRIZIONE

La funzione ***sb_reference()*** serve a produrre il puntatore a una voce della tabella dei super blocchi. Se si fornisce il numero di un dispositivo già innestato nella tabella, si intende ottenere il puntatore alla voce relativa; se si fornisce il valore zero, si intende semplicemente avere un puntatore alla prima voce (ovvero all'inizio della tabella); se invece si fornisce il valore `-1`, si vuole ottenere il riferimento alla prima voce libera.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella dei super blocchi che soddisfa la richiesta. In caso di errore, restituisce invece un puntatore nullo, ma senza dare informazioni aggiuntive con la variabile ***errno***, perché il motivo è implicito nel tipo di richiesta.

ERRORI

In caso di errore la variabile ***errno*** non viene aggiornata. Tuttavia, se l'errore deriva dalla richiesta di un dispositivo di memorizzazione, significa che non è presente nella tabella; se è stato richiesta una voce libera, significa che la tabella dei super blocchi è occupata completamente.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/sb_table.c`' [i160.4.51]

'`kernel/fs/sb_reference.c`' [i160.4.49]

VEDERE ANCHE

inode_reference(9) [i159.3.25], *file_reference(9)* [i159.3.13].

os16: sb_save(9)

NOME

'**`sb_save`**' - memorizzazione di un super blocco nel proprio file system

SINTASSI

```
<kernel/fs.h>
int sb_save (sb_t *sb);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *`***sb***	Puntatore a una voce della tabella dei super blocchi in memoria.

DESCRIZIONE

La funzione ***sb_save()*** verifica se il super blocco conservato in memoria e rappresentato dal puntatore ***sb*** risulta modificato; in tal caso provvede ad aggiornarlo nell'unità di memorizzazione di origine, assieme alle mappe di utilizzo degli inode e delle zone di dati.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il riferimento al super blocco è un puntatore nullo.
`EIO`	Errore di input-output.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/sb_save.c`' [i160.4.50]

VEDERE ANCHE

inode_save(9) [i159.3.26], *dev_io(9)* [i159.1.1].

os16: sb_zone_status(9)

Vedere *sb_inode_status(9)* [i159.3.45].

os16: stat(9)

NOME

'**`fd_stat`**', '**`path_stat`**' - interrogazione dello stato di un file

SINTASSI

```
<kernel/fs.h>
int fd_stat   (pid_t pid, int fdn, struct stat *buffer);
int path_stat (pid_t pid, const char *path,
               struct stat *buffer);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Il numero del descrittore di file.
`const char *`***path***	Il percorso che rappresenta il file.
`struct stat *`***buffer***	Area di memoria in cui scrivere le informazioni sul file.

DESCRIZIONE

Le funzioni ***fd_stat()*** e ***path_stat()*** raccolgono le informazioni disponibili sul file corrispondente al descrittore ***fdn*** del processo ***pid*** o al percorso ***path***, in una variabile strutturata di tipo '**`struct stat`**', a cui punta ***buffer***. La struttura '**`struct stat`**' è definita nel file '`lib/sys/stat.h`' nel modo seguente:

```
struct stat {
  dev_t     st_dev;     // Device containing the
                        // file.
  ino_t     st_ino;     // File serial number (inode
                        // number).
  mode_t    st_mode;    // File type and permissions.
  nlink_t   st_nlink;   // Links to the file.
  uid_t     st_uid;     // Owner user id.
  gid_t     st_gid;     // Owner group id.
  dev_t     st_rdev;    // Device number if it is a device
                        // file.
  off_t     st_size;    // File size.
  time_t    st_atime;   // Last access time.
  time_t    st_mtime;   // Last modification time.
  time_t    st_ctime;   // Last inode modification.
  blksize_t st_blksize; // Block size for I/O operations.
  blkcnt_t  st_blocks;  // File size / block size.
};
```

Va osservato che il file system Minix 1, usato da os16, riporta esclusivamente la data e l'ora di modifica, pertanto le altre due date previste sono sempre uguali a quella di modifica.

Il membro ***st_mode***, oltre alla modalità dei permessi che si cambiano con *fd_chmod(9)* [i159.3.1], serve ad annotare anche il tipo di file. Nel file '`lib/sys/stat.h`' sono definite anche delle macro-variabili per individuare e facilitare la selezione dei bit che compongono le informazioni del membro ***st_mode***:

Modalità simbolica	Modalità numerica	Descrizione
`S_IFMT`	0170000_8	Maschera che raccoglie tutti i bit che individuano il tipo di file.
`S_IFBLK`	0060000_8	File di dispositivo a blocchi.
`S_IFCHR`	0020000_8	File di dispositivo a caratteri.
`S_IFIFO`	0010000_8	File FIFO, non gestito da os16.
`S_IFREG`	0100000_8	File puro e semplice.
`S_IFDIR`	0040000_8	Directory.
`S_IFLNK`	0120000_8	Collegamento simbolico, non gestito da os16.
`S_IFSOCK`	0140000_8	Socket di dominio Unix, non gestito da os16.

Modalità simbolica	Modalità numerica	Descrizione
`S_ISUID`	0004000_8	SUID.
`S_ISGID`	0002000_8	SGID.
`S_ISVTX`	0001000_8	Sticky.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	0000700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	0000400_8	Lettura per l'utente proprietario.
`S_IWUSR`	0000200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	0000100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXG`	0000070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	0000040_8	Lettura per il gruppo.
`S_IWGRP`	0000020_8	Scrittura per il gruppo.
`S_IXGRP`	0000010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	0000007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	0000004_8	Lettura per gli altri utenti.
`S_IWOTH`	0000002_8	Scrittura per gli altri utenti.
`S_IXOTH`	0000001_8	Esecuzione per gli altri utenti.

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky; inoltre, non considera i permessi legati al gruppo, perché non tiene traccia dei gruppi.

Queste funzioni vengono usate soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento delle chiamate di sistema di tipo '**`SYS_STAT`**' e '**`SYS_FSTAT`**'. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***fd_stat()*** e ***path_stat()***, sono *fstat(2)* [u0.36] e *stat(2)* [u0.36].

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
`-1`	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Troppi file aperti nel sistema.
`ENOENT`	File non trovato.
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

FILE SORGENTI

'`lib/sys/stat/fstat.c`' [i161.13.3]

'`lib/sys/stat/stat.c`' [i161.13.6]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_stat.c`' [i160.4.11]

'`kernel/fs/path_stat.c`' [i160.4.44]

VEDERE ANCHE

fstat(2) [u0.36], *stat(2)* [u0.36], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

«

NOME

'**`zone_alloc`**', '**`zone_free`**' - allocazione di zone di dati

SINTASSI

```
<kernel/fs.h>
zno_t zone_alloc (sb_t *sb);
int   zone_free  (sb_t *sb, zno_t zone);
```


ARGOMENTI

Argomento	Descrizione
`sb_t *`***sb***	Puntatore a una voce della tabella dei super blocchi in memoria.
`zno_t` ***zone***	Numero di zona da liberare.

DESCRIZIONE

La funzione ***zone_alloc()*** occupa una zona nella mappa associata al super blocco a cui si riferisce ***sb***, restituendone il numero. La funzione ***zone_free()*** libera una zona che precedentemente risultava occupata nella mappa relativa.

VALORE RESTITUITO

La funzione ***zone_alloc()*** restituisce il numero della zona allocata. Se questo numero è zero, si tratta di un errore, e va considerato il contenuto della variabile ***errno***.

La funzione ***zone_free()*** restituisce zero in caso di successo, oppure `-1` in caso di errore, aggiornando di conseguenza la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EROFS`	Il file system è innestato in sola lettura, pertanto non è possibile apportare cambiamenti alla mappa di utilizzo delle zone.
`ENOSPC`	Non è possibile allocare una zona, perché non ce ne sono di libere.
`EINVAL`	L'argomento corrispondente a ***sb*** è un puntatore nullo; la zona di cui si richiede la liberazione è precedente alla prima zona dei dati (pertanto non può essere liberata, in quanto riguarda i dati amministrativi del super blocco); la zona da liberare è successiva allo spazio gestito dal file system.
`EUNKNOWN`	Errore imprevisto e sconosciuto.

FILE SORGENTI

'`kernel/fs.h`' [u0.4]

'`kernel/fs/zone_alloc.c`' [i160.4.53]

'`kernel/fs/zone_free.c`' [i160.4.54]

VEDERE ANCHE

zone_write(9) [i159.3.53], *sb_save(9)* [i159.3.48].

«

Vedere *zone_alloc(9)* [i159.3.51].

«

NOME

'**`zone_read`**', '**`zone_write`**' - lettura o scrittura di una zona di dati

SINTASSI

```
<kernel/fs.h>
int zone_read  (sb_t *sb, zno_t zone, void *buffer);
int zone_write (sb_t *sb, zno_t zone, void *buffer);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *`***sb***	Puntatore a una voce della tabella dei super blocchi in memoria.
`zno_t` ***zone***	Numero di zona da leggere o da scrivere
`void *`***buffer***	Puntatore alla posizione iniziale in memoria dove depositare la zona letta o da dove trarre i dati per la scrittura della zona.

DESCRIZIONE

La funzione ***zone_read()*** legge una zona e ne trascrive il contenuto a partire da ***buffer***. La funzione ***zone_write()*** scrive una

zona copiandovi al suo interno quanto si trova in memoria a partire da ***buffer***. La zona è individuata dal numero ***zone*** e riguarda il file system a cui si riferisce il super blocco ***sb***.

La lettura o la scrittura riguarda una zona soltanto, ma nella sua interezza.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EINVAL	Gli argomenti non sono validi.
EROFS	Il file system è innestato in sola lettura.
EIO	Errore di input-output.

FILE SORGENTI

‘kernel/fs.h’ [u0.4]

‘kernel/fs/zone_read.c’ [i160.4.55]

‘kernel/fs/zone_write.c’ [i160.4.56]

VEDERE ANCHE

zone_alloc(9) [i159.3.51], *zone_free(9)* [i159.3.51].

os16: ibm_i86(9)

Il file ‘kernel/ibm_i86.h’ [u0.5] descrive le funzioni e le macroistruzioni per la gestione dell’hardware.

La sezione u144 descrive complessivamente queste funzioni e le tabelle successive sono tratte da lì.

Tabella u144.2. Funzioni e macroistruzioni di basso livello, dichiarate nel file di intestazione ‘kernel/ibm_i86.h’ e descritte nei file della directory ‘kernel/ibm_i860/’. Le macroistruzioni hanno argomenti di tipo numerico non precisato, purché in grado di rappresentare il valore necessario.

Funzione o macroistruzione	Descrizione
`void _int10_00 (uint16_t video_mode);` `void int10_00 (video_mode);`	Imposta la modalità video della console. Questa funzione viene usata solo da ***con_init()***, per inizializzare la console; la modalità video è stabilita dalla macro-variabile ***IBM_I86_VIDEO_MODE***, dichiarata nel file ‘kernel/ibm_i86.h’.
`void _int10_02 (uint16_t page, uint16_t position);` `void int10_02 (page, position);`	Colloca il cursore in una posizione determinata dello schermo, relativo a una certa pagina video. Questa funzione viene usata solo da ***con_putc()***.
`void _int10_05 (uint16_t page);` `void int10_05 (page);`	Seleziona la pagina attiva del video. Questa funzione viene usata solo da ***con_init()*** e ***con_select()***.
`void _int12 (void);` `void int12 (void);`	Restituisce la quantità di memoria disponibile, in multipli di 1024 byte.

Funzione o macroistruzione	Descrizione
`void _int13_00 (uint16_t drive);` `void int13_00 (drive);`	Azzera lo stato dell’unità a disco indicata, rappresentata da un numero secondo le convenzioni del BIOS. Viene usata solo dalle funzioni ‘dsk_…()’ che si occupano dell’accesso alle unità a disco.
`uint16_t _int13_02 (uint16_t drive, uint16_t sectors, uint16_t cylinder, uint16_t head, uint16_t sector, void *buffer);` `void int13_02 (drive, sectors, cylinder, head, sector, buffer);`	Legge dei settori da un’unità a disco. Questa funzione viene usata soltanto da ***dsk_read_sectors()***.
`uint16_t _int13_03 (uint16_t drive, uint16_t sectors, uint16_t cylinder, uint16_t head, uint16_t sector, void *buffer);` `void int13_03 (drive, sectors, cylinder, head, sector, buffer);`	Scrive dei settori in un’unità a disco. Questa funzione viene usata solo da ***dsk_write_sectors()***.
`uint16_t _int16_00 (void);` `void int16_00 (void);`	Legge un carattere dalla tastiera, rimuovendolo dalla memoria tampone relativa. Viene usata solo in alcune funzioni di controllo della console, denominate ‘con_…()’.
`uint16_t _int16_01 (void);` `void int16_01 (void);`	Verifica se è disponibile un carattere dalla tastiera: se c’è ne restituisce il valore, ma senza rimuoverlo dalla memoria tampone relativa, altrimenti restituisce zero. Viene usata solo dalle funzioni di gestione della console, denominate ‘con_…()’.
`void _int16_02 (void);` `void int16_02 (void);`	Restituisce un valore con cui è possibile determinare quali funzioni speciali della tastiera risultano inserite (inserimento, fissamaiuscole, blocco numerico, ecc.). Al momento la funzione non viene usata.
`uint16_t _in_8 (uint16_t port);` `void in_8 (port);`	Legge un byte dalla porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.
`uint16_t _in_16 (uint16_t port);` `void in_16 (port);`	Legge un valore a 16 bit dalla porta di I/O indicata. Questa funzione viene usata solo da ***dev_mem()***.
`void _out_8 (uint16_t port, uint16_t value);` `void out_8 (port, value);`	Scrive un byte nella porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.
`void _out_16 (uint16_t port, uint16_t value);` `void out_16 (port, value);`	Scrive un valore a 16 bit nella porta indicata. Questa funzione viene usata solo da ***dev_mem()***.

Funzione o macroistruzione	Descrizione
`void cli (void);`	Azzera l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`CLI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void sti (void);`	Attiva l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`STI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void irq_on (unsigned int irq);`	Abilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.
`void irq_off (unsigned int irq);`	Disabilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.
`void _ram_copy (segment_t org_seg, offset_t org_off, segment_t dst_seg, offset_t dst_off, uint16_t size);` `void ram_copy (org_seg, org_off, dst_seg, dst_off, size);`	Copia una certa quantità di byte, da una posizione di memoria all'altra, specificando segmento e scostamento di origine e destinazione. Viene usata solo dalle funzioni '**`mem`**`_...()`'.

Tabella u144.3. Funzioni per l'accesso alla console, dichiarate nel file di intestazione '`kernel/ibm_i86.h`' e descritte nei file contenuti nella directory '`kernel/ibm_i86/`'.

Funzione	Descrizione
`int con_char_read (void);`	Legge un carattere dalla console, se questo è disponibile, altrimenti restituisce il valore zero. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`int con_char_wait (void);`	Legge un carattere dalla console, ma se questo non è ancora disponibile, rimane in attesa, bloccando tutto il sistema operativo. Questa funzione non è utilizzata.
`int con_char_ready (void);`	Verifica se è disponibile un carattere dalla console: se è così, restituisce un valore diverso da zero, corrispondente al carattere in attesa di essere prelevato. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`void con_init (void);`	Inizializza la gestione della console. Questa funzione viene usata solo da ***tty_init()***.
`void con_select (int console);`	Seleziona la console desiderata, dove la prima si individua con lo zero. Questa funzione viene usata solo da ***tty_console()***.

Funzione	Descrizione
`void con_putc (int console, int c);`	Visualizza il carattere indicato sullo schermo della console specificata, sulla posizione in cui si trova il cursore, facendolo avanzare di conseguenza e facendo scorrere il testo in alto, se necessario. Questa funzione viene usata solo da ***tty_write()***.
`void con_scroll (int console);`	Fa avanzare in alto il testo della console selezionata. Viene usata internamente, solo dalla funzione ***con_putc()***.

Tabella u144.6. Funzioni per l'accesso ai dischi, dichiarate nel file di intestazione '`kernel/ibm_i86.h`'.

Funzione	Descrizione
`void dsk_setup (void);`	Predispone il contenuto dell'array ***dsk_table[]***. Questa funzione viene usata soltanto da ***main()***.
`int dsk_reset (int drive);`	Azzera lo stato dell'unità corrispondente a ***dsk_table[drive].bios_drive***. Viene usata solo internamente, dalle altre funzioni '**`dsk`**`_...()`'.
`void dsk_sector_to_chs (int drive, unsigned int sector, dsk_chs_t *chs);`	Modifica le coordinate della variabile strutturata a cui punta l'ultimo parametro, con le coordinate corrispondenti al numero di settore fornito. Viene usata solo internamente, dalle altre funzioni '**`dsk`**`_...()`'.
`int dsk_read_sectors (int drive, unsigned int start_sector, void *buffer, unsigned int n_sectors);`	Legge una sequenza di settori da un disco, mettendo i dati in memoria, a partire dalla posizione espressa da un puntatore generico. La funzione è ricorsiva, ma oltre che da se stessa, viene usata internamente da ***dsk_read_bytes()*** e da ***dsk_write_bytes()***.
`int dsk_write_sectors (int drive, unsigned int start_sector, void *buffer, unsigned int n_sectors);`	Scrive una sequenza di settori in un disco, traendo i dati da un puntatore a una certa posizione della memoria. La funzione è ricorsiva, ma oltre che da se stessa, viene usata solo internamente da ***dsk_write_bytes()***.
`size_t dsk_read_bytes (int drive, off_t offset, void *buffer, size_t count);`	Legge da una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.
`size_t dsk_write_bytes (int drive, off_t offset, void *buffer, size_t count);`	Scrive su una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.

os16: k_libc(9)

Il file ‘`kernel/k_libc.h`’ [u0.6] descrive alcune funzioni con nomi che iniziano per ‘**k**_...’ (dove la lettera «k» sta per kernel) e riproducono il comportamento di funzioni standard, della libreria C. Per esempio, ***k_printf()*** è l’equivalente di ***printf()***, ma per la gestione interna del kernel.

Teoricamente, quando una funzione interna al kernel può ricondursi allo standard, dovrebbe avere il nome previsto. Tuttavia, per evitare di dover qualificare ogni volta l’ambito di una funzione, sono stati usati nomi differenti, ciò anche al fine di non creare complicazioni in fase di compilazione di tutto il sistema.

os16: main(9)

Il file ‘`kernel/main.h`’ [u0.7] descrive la funzione ***main()*** del kernel e altre funzioni accessorie, assieme al codice iniziale necessario per mettere in funzione il kernel stesso.

Si rimanda alla sezione u143 che descrive dettagliatamente il codice iniziale del kernel.

os16: memory(9)

Il file ‘`kernel/memory.h`’ [u0.8] descrive le funzioni per la gestione della memoria, a livello di sistema.

Per la descrizione dell’organizzazione della gestione della memoria si rimanda alla sezione u145. Le tabelle successive che sintetizzano l’uso delle funzioni di questo gruppo, sono tratte da quel capitolo.

Tabella u145.2. Funzioni per la gestione della mappa della memoria, dichiarate nel file di intestazione ‘`kernel/memory.h`’ e realizzate nella directory ‘`kernel/memory/`’.

Funzione	Descrizione
`uint16_t *mb_reference (void);`	Restituisce il puntatore alla tabella dei blocchi di memoria, per uniformare l’accesso alla tabella dalle funzioni che non fanno parte del gruppo contenuto nella directory ‘`kernel/memory/`’.
`ssize_t mb_alloc (addr_t` ***address***`,` `size_t` ***size***`);`	Alloca la memoria a partire dall’indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). L’allocazione ha termine anticipatamente se si incontra un blocco già utilizzato. La funzione restituisce la dimensione allocata effettivamente.
`ssize_t mb_free (addr_t` ***address***`,` `size_t` ***size***`);`	Libera la memoria a partire dall’indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). Lo spazio viene liberato in ogni caso, anche se risulta già libero; tuttavia viene prodotto un avvertimento a video se si verifica tale ipotesi.

`int mb_alloc_size`
`(size_t` ***size***`,`
`memory_t *`***allocated***`);`
	Cerca e alloca un’area di memoria della dimensione richiesta, modificando la variabile strutturata di cui viene fornito il puntatore come secondo parametro. In pratica, l’indirizzo e l’estensione della memoria allocata effettivamente si trovano nella variabile strutturata in questione, mentre la funzione restituisce zero (se va tutto bene) o `-1` se non è disponibile la memoria libera richiesta.

Tabella u145.3. Funzioni per le operazioni di lettura e scrittura in memoria, dichiarate nel file di intestazione ‘`kernel/memory.h`’ e realizzate nella directory ‘`kernel/memory/`’.

Funzione	Descrizione

`void mem_copy (addr_t` ***orig***`,`
`addr_t` ***dest***`,`
`size_t` ***size***`);`
	Copia la quantità richiesta di byte, dall’indirizzo di origine a quello di destinazione, espressi in modo efficace.

`size_t mem_read (addr_t` ***start***`,`
`void *`***buffer***`,`
`size_t` ***size***`);`
	Legge dalla memoria, a partire dall’indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene letto va poi copiato nella memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

`size_t mem_write (addr_t` ***start***`,`
`void *`***buffer***`,`
`size_t` ***size***`);`
	Scrive, in memoria, a partire dall’indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene scritto proviene dalla memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

os16: proc(9)

Il file ‘`kernel/proc.h`’ [u0.9] descrive ciò che serve per la gestione dei processi. In modo particolare, in questo file si definisce il tipo derivato ‘**`proc_t`**’, con cui si realizza la tabella dei processi.

Figura u149.19. Struttura del tipo '**proc_t**', corrispondente agli elementi dell'array ***proc_table[]***.

Listato u149.20. Struttura del tipo '**proc_t**', corrispondente agli elementi dell'array ***proc_table[]***.

```
typedef struct {
  pid_t             ppid;
  pid_t             pgrp;
  uid_t             uid;
  uid_t             euid;
  uid_t             suid;
  dev_t             device_tty;
  char              path_cwd[PATH_MAX];
  inode_t          *inode_cwd;
  int               umask;
  unsigned long int sig_status;
  unsigned long int sig_ignore;
  clock_t           usage;
  unsigned int      status;
  int               wakeup_events;
  int               wakeup_signal;
  unsigned int      wakeup_timer;
  addr_t            address_i;
  segment_t         segment_i;
  size_t            size_i;
  addr_t            address_d;
  segment_t         segment_d;
  size_t            size_d;
  uint16_t          sp;
  int               ret;
  char              name[PATH_MAX];
  fd_t              fd[FOPEN_MAX];
} proc_t;
```

Tabella u149.21. Membri del tipo '**proc_t**'.

Membro	Contenuto
`ppid`	Numero del processo genitore: *parent process id*.
`pgrp`	Numero del gruppo di processi a cui appartiene quello della voce corrispondente: *process group*. Si tratta del numero del processo a partire dal quale viene definito il gruppo.
`uid`	Identità reale del processo della voce corrispondente: *user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è stato avviato. Tuttavia, i privilegi del processo dipendono dall'identità efficace, definita dal membro '**`euid`**'.

Membro	Contenuto
`euid`	Identità efficace del processo della voce corrispondente: *effective user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è in funzione; pertanto, il processo ha i privilegi di quell'utente.
`suid`	Identità salvata: *saved user id*. Si tratta del valore che aveva ***euid*** prima di cambiare identità.
`device_tty`	Terminale di controllo, espresso attraverso il numero del dispositivo.
`path_cwd` `inode_cwd`	Entrambi i membri rappresentano la directory corrente del processo: nel primo caso in forma di percorso, ovvero di stringa, nel secondo in forma di puntatore a inode rappresentato in memoria.
`umask`	Maschera dei permessi associata al processo: i permessi attivi nella maschera vengono tolti in fase di creazione di un file o di una directory.
`sig_status`	Segnali inviati al processo e non ancora trattati: ogni segnale si associa a un bit differente del valore del membro ***sig_status***; un bit a uno indica che il segnale corrispondente è stato ricevuto e non ancora trattato.
`sig_ignore`	Segnali che il processo ignora: ogni segnale da ignorare si associa a un bit differente del valore del membro ***sig_ignore***; un bit a uno indica che quel segnale va ignorato.
`usage`	Tempo di utilizzo della CPU, da parte del processo, espresso in impulsi del temporizzatore, il quale li produce alla frequenza di circa 18,2 Hz.
`status`	Stato del processo, rappresentabile attraverso una macro-variabile simbolica, definita nel file '`proc.h`'. Per os16, gli stati possibili sono: «inesistente», quando si tratta di una voce libera della tabella dei processi; «creato», quando un processo è appena stato creato; «pronto», quando un processo è pronto per essere eseguito, «in esecuzione», quando il processo è in funzione; «sleeping», quando un processo è in attesa di qualche evento; «zombie», quando un processo si è concluso, ha liberato la memoria, ma rimangono le sue tracce perché il genitore non ha ancora recepito la sua fine.
`wakeup_events`	Eventi attesi per il risveglio del processo, ammesso che si trovi nello stato si attesa. Ogni tipo di evento che può essere atteso corrisponde a un bit e si rappresenta con una macro-variabile simbolica, dichiarata nel file '`lib/sys/os16.h`'.
`wakeup_signal`	Ammesso che il processo sia in attesa di un segnale, questo membro esprime il numero del segnale atteso.
`wakeup_timer`	Ammesso che il processo sia in attesa dello scadere di un conto alla rovescia, questo membro esprime il numero di secondi che devono ancora trascorrere.

`address_i`
`segment_i`
`size_i`
	Il valore di questi membri descrive la memoria utilizzata dal processo per le istruzioni (il segmento codice). Le informazioni sono in parte ridondanti, perché conoscendo ***segment_i*** si ottiene facilmente ***address_i*** e viceversa, ma ciò consente di ridurre i calcoli nelle funzioni che ne fanno uso.

Membro	Contenuto

`address_d`
`segment_d`
`size_d`
	Il valore di questi membri descrive la memoria utilizzata dal processo per i dati (il segmento usato per le variabili statiche e per la pila). Anche in questo caso, le informazioni sono in parte ridondanti, ma ciò consente di semplificare il codice nelle funzioni che ne fanno uso.

`sp`	Indice della pila dei dati, nell'ambito del segmento dati del processo. Il valore è significativo quando il processo è nello stato di pronto o di attesa di un evento. Quando invece un processo era attivo e viene interrotto, questo valore viene aggiornato.
`ret`	Rappresenta il valore restituito da un processo terminato e passato nello stato di «zombie».
`name`	Il nome del processo, rappresentato dal nome del programma avviato.
`fd`	Tabella dei descrittori dei file relativi al processo.

«

NOME

'**`isr_1C`**', '**`isr_80`**' - routine di gestione delle interruzioni

DESCRIZIONE

La routine '**`isr_1C`**' del file '`kernel/proc/_isr.s`' viene eseguita a ogni impulso del temporizzatore, proveniente dal sistema delle interruzioni hardware; la routine '**`isr_80`**', in modo analogo, viene eseguita in corrispondenza dell'interruzione software 80_{16} Perché ciò avvenga, nella tabella IVT, nelle voci che riguardano l'interruzione $1C_{16}$ e 80_{16}, si trova l'indirizzo corrispondente alle routine in questione. La configurazione della tabella IVT avviene per mezzo della funzione *ivt_load(9)* [i159.8.2].

La routine '**`isr_1C`**' prevede il salvataggio dei registri principali nella pila dei dati in funzione al momento dell'interruzione. Quindi vengono modificati i registri che definiscono l'area dati (ES e DS) e successivamente ciò permette di intervenire su delle variabili locali: viene incrementato il contatore degli impulsi del temporizzatore; viene incrementato il contatore dei secondi, se il contatore degli impulsi è divisibile per 18 senza dare resto; vengono salvati l'indice e il segmento della pila dei dati, in due variabili locali.

Dalla verifica del valore del segmento in cui si colloca la pila dei dati del processo interrotto, la routine verifica se si tratta di un processo comune o del kernel. Se si tratta di un processo comune, si scambia la pila con quella del kernel. Per questo la routine si avvale della variabile ***_ksp*** (*kernel stack pointer*), usata anche dalla funzione *proc_scheduler(9)* [i159.8.11]. Sempre se si tratta dell'interruzione di un processo diverso dal kernel, viene chiamata la funzione ***proc_scheduler()***, già citata, fornendo come argomenti il puntatore alla variabile che contiene l'indice della pila e il puntatore alla variabile che contiene il segmento di memoria che ospita la pila dei dati. Al termine viene scambiata nuovamente la pila dei dati, usando come valori quanto contenuto nelle variabili che prima sono servite per salvare l'indice e il segmento della pila.

Poi, indipendentemente dal tipo di processo, vengono ripristinati i registri accumulati in precedenza nella pila e viene restituito il controllo, concludendo il lavoro dell'interruzione.

Va osservato che la funzione ***proc_scheduler()*** riceve l'indice e il segmento della pila dei dati attraverso dei puntatori a variabili scalari. Pertanto, tale funzione è perfettamente in grado di sostituire questi valori, con quelli della pila di un altro processo. Per questo, quando al ritorno della funzione viene ripristinata la pila sulla base di tali variabili, si ha uno scambio di processi. Il ripristino successivo dalla pila dei registri, completa il procedimento di sostituzione dei processi.

La routine '**`isr_80`**' viene attivata da un'interruzione software, dovuta a una chiamata di sistema. Questa routine si distingue leggermente da '**`isr_1C`**', in quanto non si occupa di tenere conto del tempo trascorso, ma ha la necessità di recuperare dalla pila del processo interrotto, i valori che hanno origine dalla chiamata di sistema. Si tratta sempre del numero della chiamata di sistema, del puntatore al messaggio trasmesso con la chiamata e della sua lunghezza.

Si può osservare anche un'altra differenza importante, per cui, se l'interruzione riguarda il processo del kernel, l'indice della pila dello stesso viene conservato nella variabile ***_ksp***. Questo fatto è importante, perché prima di abilitare la gestione delle interruzioni, è necessario che il kernel stesso ne provochi una, in modo da poter salvare la prima volta l'indice della propria pila.

Successivamente, indipendentemente dal processo interrotto, si chiama la funzione *sysroutine(9)* [i159.8.28], alla quale si passano come argomenti, oltre che i puntatori all'indice e al segmento della pila dei dati del processo interrotto, anche gli argomenti della chiamata di sistema.

La funzione ***sysroutine()*** si avvale a sua volta della funzione ***proc_scheduler()***, pertanto anche in questo caso la pila dei dati che viene ripristinata successivamente può risultare differente da quella del processo interrotto originariamente, comportando anche in questo caso lo scambio del processo con un altro.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/_isr.s`' [i160.9.1]

VEDERE ANCHE

ivt_load(9) [i159.8.2], *sys(2)* [u0.37], *proc_scheduler(9)* [i159.8.11], *sysroutine(9)* [i159.8.28].

«

NOME

'**`ivt_load`**' - caricamento della tabella IVT

SINTASSI

```
<kernel/proc.h>
void _ivt_load (void);
```

```
<kernel/proc.h>
void  ivt_load (void);
```

DESCRIZIONE

La funzione ***_ivt_load()***, ovvero la macroistruzione corrispondente ***ivt_load()***, modifica la tabella IVT del BIOS, in modo che nella posizione corrispondente all'interruzione $1C_{16}$ ci sia il puntatore alla routine *isr_1C(9)* [i159.8.1], e che in corrispondenza dell'interruzione 80_{16} ci sia il puntatore alla routine *isr_80(9)* [i159.8.1].

Questa funzione viene usata una volta sola, all'interno di *main(9)* [u0.6].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/_ivt_load.s`' [i160.9.2]

VEDERE ANCHE

sys(2) [u0.37], *isr_80(9)* [i159.8.1], *proc_scheduler(9)* [i159.8.11], *sysroutine(9)* [i159.8.28].

NOME

'**proc_available**' - inizializzazione di un processo libero

SINTASSI

```
<kernel/proc.h>
void proc_available (pid_t pid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo da inizializzare.

DESCRIZIONE

La funzione ***proc_available()*** si limita a inizializzare, con valori appropriati, i dati di un processo nella tabella relativa, in modo che risulti correttamente uno spazio libero per le allocazioni successive.

Questa funzione viene usata da *proc_init(9)* [i159.8.6], *proc_sig_chld(9)* [i159.8.12], *proc_sys_wait(9)* [i159.8.27].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/proc_available.c`' [i160.9.3]

NOME

'**proc_dump_memory**' - copia di una porzione di memoria in un file

SINTASSI

```
<kernel/proc.h>
void proc_dump_memory (pid_t pid, addr_t address, size_t size,
                       char *name);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`addr_t` ***address***	Indirizzo efficace della memoria.
`size_t` ***size***	Quantità di byte da trascrivere, a partire dall'indirizzo efficace.
`char *`***name***	Nome del file da creare.

DESCRIZIONE

La funzione ***proc_dump_memory()*** salva in un file una porzione di memoria, secondo le coordinate fornita dagli argomenti.

Viene usata esclusivamente da *proc_sig_core(9)* [i159.8.6], quando si riceve un segnale per cui è necessario scaricare la memoria di un processo. In quel caso, se il processo eliminato ha i permessi per scrivere nella directory radice, vengono creati due file: uno con l'immagine del segmento codice ('`/core.i`') e l'altro con l'immagine del segmento dati ('`/core.d`').

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_core.c`' [i160.9.14]

VEDERE ANCHE

fd_open(9) [i159.3.8], *fd_write(9)* [i159.3.12], *fd_close(9)* [i159.3.3].

NOME

'**proc_find**' - localizzazione di un processo sulla base dell'indirizzo del segmento dati

SINTASSI

```
<kernel/proc.h>
pid_t proc_find (segment_t segment_d);
```

ARGOMENTI

Argomento	Descrizione
`segment_t` ***segment_d***	Indirizzo del segmento da cercare nella tabella dei processi, come allocato per il segmento dati.

DESCRIZIONE

La funzione ***proc_find()*** scandisce la tabella dei processi, alla ricerca di quel processo il cui segmento dati corrisponde al valore fornito come argomento. Ciò serve per sapere chi sia il processo interrotto, del quale si conosce il valore che, prima dell'interruzione, aveva il registro DS (*data segment*).

Questa funzione viene usata da *proc_scheduler(9)* [i159.8.11] e da *sysroutine(9)* [i159.8.28].

VALORE RESTITUITO

La funzione restituisce il numero del processo trovato e non è ammissibile che la ricerca possa fallire. Infatti, se così fosse, si produrrebbe un errore fatale, con avvertimento a video, tale da arrestare il funzionamento del kernel.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/proc_find.c`' [i160.9.5]

NOME

'**proc_init**' - inizializzazione della gestione complessiva dei processi elaborativi

SINTASSI

```
<kernel/proc.h>
extern uint16_t _etext;
void proc_init (void);
```

ARGOMENTI

Argomento	Descrizione
`extern uint16_t` ***_etext;***	La variabile ***_etext*** viene fornita dal compilatore e rappresenta l'indirizzo in cui l'area codice si è conclusa. Il valore di ***_etext*** si riferisce a un'area codice che inizia dall'indirizzo zero, pertanto questo dato viene usato per conoscere la dimensione dell'area codice del kernel.

DESCRIZIONE

La funzione ***proc_init()*** viene usata una volta sola, dalla funzione *main(9)* [u0.6], per predisporre la gestione dei processi. Per la precisione svolge le operazioni seguenti:

- carica la tabella IVT, in modo che le interruzioni software $1C_{16}$ e 80_{16} siano dirette correttamente al codice che deve gestirle;
- programma il temporizzatore interno, in modo da produrre una frequenza di circa 18,2 Hz;
- inizializza la tabella dei processi in modo che tutti gli alloggiamenti previsti risultino liberi;
- innesta il file system principale, presupponendo che possa trattarsi soltanto della prima unità a dischetti;

- inizializza correttamente le voci del processo zero, ovvero quelle del kernel, segnando anche come allocata la porzione di memoria utilizzata dal kernel e lo spazio iniziale usato dal BIOS (tabella IVT e BDA);
- abilita le interruzioni hardware del temporizzatore, della tastiera e dell'unità a dischetti: le altre interruzioni hardware rimangono disabilitate.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/proc_init.c`' [i160.9.6]

VEDERE ANCHE

ivt_load(9) [i159.8.2], *proc_available(9)* [i159.8.3], *sb_mount(9)* [i159.3.46].

os16: proc_reference(9)

NOME

'**`proc_reference`**' - puntatore alla voce che rappresenta un certo processo

SINTASSI

```
<kernel/proc.h>
proc_t *proc_reference (pid_t pid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo cercato nella tabella relativa.

DESCRIZIONE

La funzione ***proc_reference()*** serve a produrre il puntatore all'elemento dell'array ***proc_table[]*** che contiene i dati del processo indicato per numero come argomento.

Viene usata dalle funzioni che non fanno parte del gruppo di '`kernel/proc.h`'.

VALORE RESTITUITO

Restituisce il puntatore all'elemento della tabella ***proc_table[]*** che rappresenta il processo richiesto. Se il numero del processo richiesto non può esistere, la funzione restituisce il puntatore nullo '**`NULL`**'.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/proc_reference.c`' [i160.9.7]

os16: proc_sch_signals(9)

NOME

'**`proc_sch_signals`**' - verifica dei segnali dei processi

SINTASSI

```
<kernel/proc.h>
void proc_sch_signals (void);
```

DESCRIZIONE

La funzione ***proc_sch_signals()*** ha il compito di scandire tutti i processi della tabella ***proc_table[]***, per verificare lo stato di attivazione dei segnali e procedere di conseguenza.

Dal punto di vista pratico, la funzione si limita a scandire i numeri PID possibili, demandando ad altre funzioni il compito di fare qualcosa nel caso fosse attivato l'indicatore di un segnale. Va comunque osservato che os16 si limita a gestire le azioni predefinite, pertanto si può soltanto attivare o inibire i segnali, salvo i casi in cui questi non possono essere mascherati.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

'`kernel/proc/proc_sch_signals.c`' [i160.9.8]

VEDERE ANCHE

proc_sig_term(9) [i159.8.19], *proc_sig_core(9)* [i159.8.14], *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_stop(9)* [i159.8.18].

os16: proc_sch_terminals(9)

NOME

'**`proc_sch_terminals`**' - acquisizione di un carattere dal terminale attivo

SINTASSI

```
<kernel/proc.h>
void proc_sch_terminals (void);
```

DESCRIZIONE

La funzione ***proc_sch_terminals()*** ha il compito di verificare la presenza di un carattere digitato dalla console. Se verifica che effettivamente è stato digitato un carattere, dopo aver determinato a quale terminale virtuale si riferisce, determina se per quel terminale era già stato accumulato un carattere, e se è effettivamente così, sovrascrive quel carattere ma annota anche che l'inserimento precedente è stato perduto.

Successivamente verifica se quel terminale virtuale è associato a un gruppo di processi; se è così e se il carattere corrisponde alla combinazione [*Ctrl c*], invia il segnale SIGINT a tutti i processi di quel gruppo, ma senza poi accumulare il carattere.

Indipendentemente dal fatto che il terminale appartenga a un gruppo di processi, controlla che il carattere inserito sia stato ottenuto, rispettivamente, con le combinazioni di tasti [*Ctrl q*], [*Ctrl r*], [*Ctrl s*] e [*Ctrl t*], nel qual caso attiva la console virtuale relativa (dalla prima alla quarta), evitando di accumulare il carattere.

Alla fine, scandisce tutti i processi sospesi in attesa di input dal terminale, risvegliandoli (ogni processo deve poi verificare se effettivamente c'è un carattere per sé oppure no, e se non c'è dovrebbe rimettersi in attesa).

Questa funzione viene usata soltanto da *proc_scheduler(9)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

'`kernel/proc/proc_sch_terminals.c`' [i160.9.9]

os16: proc_sch_timers(9)

NOME

'**`proc_sch_timers`**' - verifica dell'incremento del contatore del tempo

SINTASSI

```
<kernel/proc.h>
void proc_sch_timers (void);
```

DESCRIZIONE

La funzione ***proc_sch_timers()*** verifica che il calendario si sia incrementato di almeno una unità temporale (per os16 è un secondo soltanto) e se è così, va a risvegliare tutti i processi sospesi in attesa del passaggio di un certo tempo. Tali processi, una volta messi effettivamente in funzione, devono verificare che sia trascorsa effettivamente la quantità di tempo desiderata, altrimenti devono rimettersi a riposo in attesa del tempo rimanente.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

'`kernel/proc/proc_sch_timers.c`' [i160.9.10]

os16: proc_scheduler(9)

NOME

'**`proc_scheduler`**' - schedulatore

SINTASSI

```
<kernel/proc.h>
void proc_scheduler (uint16_t *sp, segment_t *segment_d);
```

ARGOMENTI

Argomento	Descrizione
`extern uint16_t` ***_ksp***	L'indice della pila del kernel.
`uint16_t` ****sp***	Puntatore all'indice della pila del processo interrotto.
`segment_t` ****segment_d***	Puntatore al segmento dati del processo interrotto.

DESCRIZIONE

La funzione ***proc_scheduler()*** viene avviata a seguito di un'interruzione hardware, dovuta al temporizzatore, oppure a seguito di un'interruzione software, dovuta a una chiamata di sistema.

La funzione determina qual è il processo interrotto, scandendo la tabella dei processi alla ricerca di quello il cui segmento dati corrisponde al valore ***segment_d***. Per questo si avvale di *proc_find(9)* [i159.8.5].

Successivamente verifica se ci sono processi in attesa di un evento del temporizzatore o del terminale, inoltre verifica se ci sono processi con segnali in attesa di essere presi in considerazione. per fare questo si avvale di *proc_sch_timers(9)* [i159.8.10], *proc_sch_terminals(9)* [i159.8.9] e *proc_sch_signals(9)* [i159.8.8], che provvedono a fare ciò che serve in presenza degli eventi di propria competenza.

Si occupa quindi di annotare il tempo di CPU utilizzato dal processo appena sospeso, misurato in unità di tempo a cui si riferisce il tipo '**`clock_t`**'.

Successivamente scandisce la tabella dei processi alla ricerca di un altro processo da mettere in funzione, al posto di quello sospeso. Se trova un processo pronto per questo lo elegge a processo attivo, declassando quello sospeso a processo pronto ma in attesa, inoltre aggiorna i valori per le variabili ****sp*** e ****segment_d***.

Al termine salva nella variabile globale ***_ksp*** il valore dell'indice della pila del kernel, come appare nelle informazioni della tabella dei processi e poi manda il messaggio «EOI» (*end of interrupt* al «PIC 1» (*programmable interrupt controller*).

Questa funzione viene usata dalla routine *isr_1C(9)* [i159.8.1] del file '`kernel/proc/_isr.s`' e dalla funzione *sysroutine(9)* [i159.8.28].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

VEDERE ANCHE

proc_find(9) [i159.8.5], *proc_sch_timers(9)* [i159.8.10], *proc_sch_signals(9)* [i159.8.8], *proc_sch_terminals(9)* [i159.8.9].

os16: proc_sig_chld(9)

NOME

'**`proc_sig_chld`**' - procedura associata alla ricezione di un segnale SIGCHLD

SINTASSI

```
<kernel/proc.h>
void proc_sig_chld (pid_t parent, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***parent***	Numero del processo considerato, il quale potrebbe avere ricevuto un segnale SIGCHLD.
`int` ***sig***	Numero del segnale: deve trattarsi esclusivamente di quanto corrispondente a SIGCHLD.

DESCRIZIONE

La funzione ***proc_sig_chld()*** si occupa di verificare che il processo specificato con il parametro ***parent*** abbia ricevuto precedentemente un segnale SIGCHLD. Se risulta effettivamente così, allora va a verificare se tale segnale risulta ignorato per quel processo: se è preso in considerazione verifica ancora se quel processo è sospeso proprio in attesa di un segnale SIGCHLD. Se si tratta di un processo che sta attendendo tale segnale, allora viene risvegliato, altrimenti, sempre ammesso che comunque il segnale non sia ignorato, la funzione elimina tutti i processi figli di ***parent***, i quali risultano già defunti, ma non ancora rimossi dalla tabella dei processi (pertanto processi «zombie»).

In pratica, se il processo ***parent*** sta attendendo un segnale SIGCHLD, significa che al risveglio si aspetta di verificare la morte di uno dei suoi processi figli, in modo da poter ottenere il valore di uscita con cui questo si è concluso. Diversamente, non c'è modo di informare il processo ***parent*** di tali conclusioni, per cui a nulla servirebbe continuare a mantenerne le tracce nella tabella dei processi.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_chld.c`' [i160.9.12]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

os16: proc_sig_cont(9)

NOME

'**`proc_sig_cont`**' - ripresa di un processo sospeso in attesa di qualcosa

SINTASSI

```
<kernel/proc.h>
void proc_sig_cont (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi esclusivamente di quanto corrispondente a SIGCONT.

DESCRIZIONE

La funzione ***proc_sig_cont()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale SIGCONT e che questo non sia stato disabilitato. In tal caso, assegna al processo lo status di «pronto» (‘**`PROC_READY`**’), ammesso che non si trovasse già in questa situazione.

Lo scopo del segnale SIGCONT è quindi quello di far riprendere un processo che in precedenza fosse stato sospeso attraverso un segnale SIGSTOP, SIGTSTP, SIGTTIN oppure SIGTTOU.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_sig_cont.c`’ [i160.9.13]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

os16: proc_sig_core(9)

NOME

‘**`proc_sig_core`**’ - chiusura di un processo e scarico della memoria su file

SINTASSI

```
<kernel/proc.h>
void proc_sig_core (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di un segnale a cui si associa in modo predefinito la conclusione e lo scarico della memoria.

DESCRIZIONE

La funzione ***proc_sig_core()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale tale da richiedere la conclusione e lo scarico della memoria del processo stesso, e che il segnale in questione non sia stato disabilitato. In tal caso, la funzione chiude il processo, ma prima ne scarica la memoria su uno o due file, avvalendosi per questo della funzione *proc_dump_memory(9)* [i159.8.4].

Un segnale atto a produrre lo scarico della memoria, potrebbe essere prodotto anche a seguito di un errore rilevato dalla CPU, come una divisione per zero. Tuttavia, il kernel di os16 non riesce a intrappolare errori di questo tipo, dato che dalla tabella IVT vengono presi in considerazione soltanto l’impulso del temporizzatore e le chiamate di sistema. In altri termini, se un programma produce effettivamente un errore così grave da essere rilevato dalla CPU, al sistema operativo non arriva alcuna comunicazione. Pertanto, tali segnali possono essere soltanto provocati deliberatamente.

Lo scarico della memoria, nell’eventualità di un errore così grave, dovrebbe servire per consentire un’analisi dello stato del processo nel momento del verificarsi di un errore fatale. Sotto questo aspetto, va anche considerato che l’area dati dei processi è priva

di etichette che possano agevolare l’interpretazione dei contenuti e, di conseguenza, non ci sono strumenti che consentano tale attività.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_sig_core.c`’ [i160.9.14]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16], *proc_dump_memory(9)* [i159.8.4].

os16: proc_sig_ignore(9)

NOME

‘**`proc_sig_ignore`**’ - verifica dello stato di inibizione di un segnale

SINTASSI

```
<kernel/proc.h>
int proc_sig_ignore (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da verificare.

DESCRIZIONE

La funzione ***proc_sig_ignore()*** verifica se, per un certo processo ***pid***, il segnale ***sig*** risulti inibito.

Questa funzione viene usata da *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19], per verificare se un segnale sia stato inibito, prima di applicarne le conseguenze, nel caso fosse stato ricevuto.

VALORE RESTITUITO

Valore	Significato
1	Il segnale risulta bloccato (inibito).
0	Il segnale è abilitato regolarmente.

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_sig_ignore.c`’ [i160.9.15]

VEDERE ANCHE

proc_sig_chld(9) [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18]m *proc_sig_term(9)* [i159.8.19].

os16: proc_sig_on(9)

NOME

‘**`proc_sig_on`**’, ‘**`proc_sig_off`**’ - registrazione o cancellazione di un segnale per un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_on  (pid_t pid, int sig);
void proc_sig_off (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da registrare o da cancellare.

DESCRIZIONE

La funzione ***proc_sig_on()*** annota per il processo ***pid*** la ricezione del segnale ***sig***; la funzione ***proc_sig_off()*** procede invece in senso opposto, cancellando quel segnale.

La funzione ***proc_sig_off()*** viene usata quando l'azione prevista per un segnale che risulta ricevuto è stata eseguita, allo scopo di riportare l'indicatore di quel segnale in una condizione di riposo. Si tratta delle funzioni *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19].

La funzione ***proc_sig_on()*** viene usata quando risulta acquisito un segnale o quando il contesto lo deve produrre, per annotarlo. Si tratta delle funzioni *proc_sys_exit(9)* [i159.8.21] e *proc_sys_kill(9)* [i159.8.23].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_on.c`' [i160.9.17]

'`kernel/proc/proc_sig_off.c`' [i160.9.16]

VEDERE ANCHE

proc_sys_exit(9) [i159.8.21], *proc_sys_kill(9)* [i159.8.23], *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18], *proc_sig_term(9)* [i159.8.19].

os16: proc_sig_status(9)

NOME

'**`proc_sig_status`**' - verifica dello stato di ricezione di un segnale

SINTASSI

```
<kernel/proc.h>
int proc_sig_status (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da verificare.

DESCRIZIONE

La funzione ***proc_sig_status()*** verifica se, per un certo processo ***pid***, il segnale ***sig*** risulti essere stato ricevuto (registrato).

Questa funzione viene usata da *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19], per verificare se un segnale è stato ricevuto effettivamente, prima di applicarne eventualmente le conseguenze.

VALORE RESTITUITO

Valore	Significato
1	Il segnale risulta ricevuto.
0	Il segnale risulta cancellato.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_status.c`' [i160.9.18]

VEDERE ANCHE

proc_sig_chld(9) [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18], *proc_sig_term(9)* [i159.8.19].

os16: proc_sig_stop(9)

NOME

'**`proc_sig_stop`**' - sospensione di un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_stop (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU.

DESCRIZIONE

La funzione ***proc_sig_stop()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU, e che questo non sia stato disabilitato. In tal caso, sospende il processo, lasciandolo in attesa di un segnale (SIGCONT).

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_stop.c`' [i160.9.19]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

os16: proc_sig_term(9)

NOME

'**`proc_sig_term`**' - conclusione di un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_term (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di un segnale per cui si associa la conclusione del processo, ma senza lo scarico della memoria.

DESCRIZIONE

La funzione ***proc_sig_term()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale per cui si prevede generalmente la conclusione del processo. Inoltre, la funzione verifica che il segnale non sia stato inibito, con l'eccezione che per il segnale SIGKILL un'eventuale inibizione non viene considerata (in quanto segnale non mascherabile). Se il segnale risulta ricevuto e valido, procede con la conclusione del processo.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_term.c`' [i160.9.20]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16], *proc_sys_exit(9)* [i159.8.21].

NOME

'**proc_sys_exec**' - sostituzione di un processo esistente con un altro, ottenuto dal caricamento di un file eseguibile

SINTASSI

```
<kernel/proc.h>
int proc_sys_exec (uint16_t *sp, segment_t *segment_d,
                   pid_t pid, const char *path,
                   unsigned int argc, char *arg_data,
                   unsigned int envc, char *env_data);
```

ARGOMENTI

Argomento	Descrizione
`uint16_t *sp`	Puntatore alla variabile contenente l'indice della pila dei dati del processo che ha eseguito la chiamata di sistema *execve(2)* [u0.10].
`segment_t *segment_d`	Puntatore alla variabile contenente il valore del segmento dati del processo che ha eseguito la chiamata di sistema *execve(2)* [u0.10].
`pid_t pid`	Il numero del processo corrispondente.
`const char *path`	Il percorso assoluto del file da caricare ed eseguire.
`unsigned int argc`	La quantità di argomenti per l'avvio del nuovo processo, incluso il nome del processo stesso.
`char *arg_data`	Una sequenza di stringhe (dopo la terminazione di una inizia la successiva), ognuna contenente un argomento da passare al processo.
`unsigned int envc`	La quantità di variabili di ambiente da passare al nuovo processo.
`char *env_data`	Una sequenza di stringhe (dopo la terminazione di una inizia la successiva), ognuna contenente l'assegnamento di una variabile di ambiente.

I parametri ***arg_data*** e ***env_data*** sono stringhe multiple, nel senso che sono separate le une dalle altre dal codice nullo di terminazione. Per sapere quante sono effettivamente le stringhe da cercare a partire dai puntatori che costituiscono effettivamente questi due parametri, si usano ***argc*** e ***envc***.

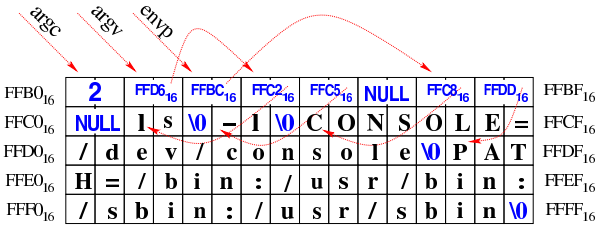
DESCRIZIONE

La funzione ***proc_sys_exec()*** serve a mettere in pratica la chiamata di sistema *execve(2)* [u0.10], destinata a rimpiazzare il processo in corso con un nuovo processo, caricato da un file eseguibile.

La funzione ***proc_sys_exec()***, dopo aver verificato che si tratti effettivamente di un file eseguibile valido e che ci siano i permessi per metterlo in funzione, procede all'allocazione della memoria, dividendo se necessario l'area codice da quella dei dati, quindi legge il file e copia opportunamente le componenti di questo nelle aree di memoria allocate.

Terminato il caricamento del file, viene ricostruita in memoria la pila dei dati del nuovo processo. Prima si mettono sul fondo le stringhe delle variabili di ambiente e quelle degli argomenti della chiamata, quindi si aggiungono i puntatori alle stringhe delle variabili di ambiente, ricostruendo così l'array noto convenzionalmente come '**envp[]**', continuando con l'aggiunta dei puntatori alle stringhe degli argomenti della chiamata, per riprodurre l'array '**argv[]**'. Per ricostruire gli argomenti della chiamata della funzione ***main()*** dell'applicazione, vanno però aggiunti ancora: il puntatore all'inizio dell'array delle stringhe che descrivono le variabili di ambiente, il puntatore all'array delle stringhe che descrivono gli argomenti della chiamata e il valore che rappresenta la quantità di argomenti della chiamata.

Figura u159.145. Caricamento degli argomenti della chiamata della funzione ***main()***.



Fatto ciò, vanno aggiunti tutti i valori necessari allo scambio dei processi, costituiti dai vari registri da rimpiazzare.

Superato il problema della ricostruzione della pila dei dati, la funzione ***proc_sys_exec()*** predispone i descrittori di standard input, standard output e standard error, quindi libera la memoria usata dal processo chiamante e ne rimpiazza i dati nella tabella dei processi con quelli del nuovo processo caricato.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_EXEC**'.

FILE SORGENTI

'`lib/unistd/execve.c`' [i161.17.13]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_exec.c`' [i160.9.21]

VEDERE ANCHE

execve(2) [u0.10], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *path_inode(9)* [i159.3.36], *inode_check(9)* [i159.3.16], *inode_put(9)* [i159.3.24], *inode_file_read(9)* [i159.3.18], *dev_io(9)* [i159.1.1], *fd_close(9)* [i159.3.3].

NOME

'**proc_sys_exit**' - chiusura di un processo elaborativo

SINTASSI

```
<kernel/proc.h>
void proc_sys_exit (pid_t pid, int status);
```

ARGOMENTI

Argomento	Descrizione
`pid_t pid`	Il numero del processo elaborativo da concludere.
`int status`	Il valore di uscita del processo da concludere.

DESCRIZIONE

La funzione ***proc_sys_exit()*** conclude il processo indicato come argomento, chiudendo tutti i descrittori di file che risultano ancora aperti e liberando la memoria. Precisamente compie i passaggi seguenti:

- aggiorna la tabella dei processi indicando per questo lo stato di «zombie» e annotando il valore di uscita;
- chiude i descrittori di file che risultano aperti;
- chiude l'inode della directory corrente;
- se si tratta del processo principale di un gruppo di processi, allora chiude anche il terminale di controllo;
- libera la memoria utilizzata dal processo, verificando comunque che l'area usata per il codice non sia abbinata anche a un altro processo, nel qual caso l'area del codice verrebbe preservata;

- se ci sono dei processi figli di quello che si va a chiudere, questi vengono abbandonati e affidati al processo numero uno (‘**`init`**’);
- se sono stati abbandonati dei processi, invia il segnale SIGCHLD al processo numero uno (‘**`init`**’);
- invia al processo genitore il segnale, in modo che possa valutare, eventualmente, il valore di uscita del processo ormai defunto.

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_EXIT`**’, e anche dalle funzioni *proc_sig_core(9)* [i159.8.14] e *proc_sig_term(9)* [i159.8.19].

FILE SORGENTI

‘`lib/unistd/_exit.c`’ [i161.17.1]

‘`lib/stdlib/_Exit.c`’ [i161.10.1]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/proc/proc_sys_exit.c`’ [i160.9.22]

VEDERE ANCHE

_exit(2) [u0.2], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sig_core(9)* [i159.8.14], *proc_sig_term(9)* [i159.8.19], *fd_close(9)* [i159.3.3], *inode_put(9)* [i159.3.24], *proc_sig_on(9)* [i159.8.16].

os16: proc_sys_fork(9)

NOME

‘**`proc_sys_fork`**’ - sdoppiamento di un processo elaborativo

SINTASSI

```
<kernel/proc.h>
pid_t proc_sys_fork (pid_t ppid, uint16_t sp);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***ppid***	Il numero del processo che chiede di creare un figlio uguale a se stesso.
`uint16_t` ***sp***	Indice della pila del processo da duplicare.

DESCRIZIONE

La funzione ***proc_sys_fork()*** crea un duplicato del processo chiamante, il quale diventa figlio dello stesso. Precisamente, la funzione compie i passaggi seguenti:

- cerca un alloggiamento libero nella tabella dei processi e procede solo se questo risulta disponibile effettivamente;
- per sicurezza, analizza i processi che risultano essere defunti (zombie) e ne inizializza i valori delle allocazioni in memoria;
- alloca la memoria necessaria a ottenere la copia del processo, tenendo conto che se il processo originario divide l’area codice da quella dei dati, è necessario allocare soltanto lo spazio per l’area dati, in quanto quella del codice può essere condivisa (essendo usata soltanto in lettura);
- compila le informazioni necessarie nella tabella dei processi, relative al nuovo processo da produrre, dichiarandolo come figlio di quello chiamante;
- incrementa il contatore di utilizzo dell’inode che rappresenta la directory corrente, in quanto un nuovo processo la va a utilizzare;
- duplica i descrittori di file già aperti per il processo da duplicare, incrementando di conseguenza il contatore dei riferimenti nella tabella dei file;

- modifica i valori dei registri di segmento nella pila dei dati riferita al processo nuovo, per renderli coerenti con la nuova collocazione in memoria;
- mette il nuovo processo nello stato di pronto, annotandolo così nella tabella dei processi;
- restituisce il numero del nuovo processo: nel processo figlio, invece, non restituisce alcunché.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**`SYS_FORK`**’.

VALORE RESTITUITO

La funzione restituisce al processo chiamante il numero del processo figlio, mentre il risultato che si ottiene nel processo figlio che si trova a riprendere il funzionamento dallo stesso punto, è semplicemente zero. Ciò consente di distinguere quale sia il processo genitore e quale è invece il figlio. Se la funzione non è in grado di portare a termine il lavoro di duplicazione dei processi, restituisce il valore `-1`, aggiornando di conseguenza la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Non c’è memoria sufficiente, oppure la tabella dei processi è occupata completamente.

FILE SORGENTI

‘`lib/unistd/fork.c`’ [i161.17.17]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/proc/proc_sys_fork.c`’ [i160.9.23]

VEDERE ANCHE

fork(2) [u0.14], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *dev_io(9)* [i159.1.1].

os16: proc_sys_kill(9)

NOME

‘**`proc_sys_kill`**’ - invio di un segnale a uno o più processi elaborativi

SINTASSI

```
<kernel/proc.h>
int proc_sys_kill (pid_t pid_killer, pid_t pid_target, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid_killer***	Il numero del processo per conto del quale si invia il segnale.
`pid_t` ***pid_target***	Il numero del processo che dovrebbe ricevere il segnale.
`int` ***sig***	Il numero del segnale da inviare.

DESCRIZIONE

La funzione ***proc_sys_kill()*** invia il segnale ***sig*** al processo numero ***pid_target***, ammesso che il processo ***pid_killer*** abbia i privilegi necessari a farlo. Tuttavia, se il numero ***pid_target*** è zero o `-1`, si richiede alla funzione l’invio del segnale a un insieme di processi. La tabella successiva descrive i vari casi.

Identità efficace del processo ***pid_killer***	Valore di ***pid_target***	Effetto.
--	< -1	Il valore di ***pid_target*** non è ammissibile: si ottiene un errore.
0	-1	Viene inviato il segnale ***sig*** a tutti i processi con UID ≥ 2 (si esclude il kernel e il processo numero uno, '**init**').
> 0	-1	Viene inviato il segnale ***sig*** a tutti i processi con UID ≥ 1 (si esclude il kernel) la cui identità efficace coincide con quella di ***pid_killer***.
--	0	Viene inviato il segnale ***sig*** a tutti i processi che appartengono allo stesso gruppo di ***pid_target***.
--	> 0	Viene inviato il segnale ***sig*** al processo ***pid_target***, purché l'identità reale o efficace del processo ***pid_killer*** sia uguale all'identità reale o salvata del processo ***pid_target***.

Si osservi che il preteso invio di un segnale pari a zero, ovvero di un segnale nullo, non produce alcun effetto, ma la funzione segnala comunque di avere completato l'operazione con successo.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_KILL**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
ESRCH	Il processo ***pid_target*** non esiste, non è un processo che possa ricevere segnali, oppure il valore dato non è interpretabile in alcun modo.
EPERM	Il processo ***pid_killer*** non ha i privilegi necessari a inviare il segnale a ***pid_target***.

FILE SORGENTI

'lib/signal/kill.c' [i161.8.1]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_kill.c' [i160.9.24]

VEDERE ANCHE

kill(2) [u0.22], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sig_on(9)* [i159.8.16].

os16: proc_sys_seteuid(9)

NOME

'**proc_sys_seteuid**' - modifica dell'identità efficace

SINTASSI

```
<kernel/proc.h>
int proc_sys_seteuid (pid_t pid, uid_t euid);
```


ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Processo su cui intervenire per il cambiamento di identità efficace.
uid_t ***euid***	Nuova identità efficace richiesta.

DESCRIZIONE

La funzione ***proc_sys_seteuid()*** modifica l'identità efficace del processo ***pid***, purché si verifichino certe condizioni:

- se il processo ***pid*** è zero, l'identità efficace viene modificata senza altre verifiche;
- se l'identità efficace che ha già il processo coincide con quella nuova richiesta, non viene apportata alcuna modifica (per ovvi motivi);
- se la nuova identità efficace corrisponde all'identità reale del processo, oppure se corrisponde alla sua identità salvata, allora la modifica di quella efficace ha luogo come richiesto;
- in tutti gli altri casi si ottiene un errore.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_SETEUID**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EPERM	Il processo ***pid*** non può cambiare l'identità efficace con il valore richiesto.

FILE SORGENTI

'lib/unistd/seteuid.c' [i161.17.30]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_seteuid.c' [i160.9.25]

VEDERE ANCHE

seteuid(2) [u0.33], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_setuid(9)* [i159.8.25].

os16: proc_sys_setuid(9)

NOME

'**proc_sys_setuid**' - modifica dell'identità

SINTASSI

```
<kernel/proc.h>
int proc_sys_setuid (pid_t pid, uid_t uid);
```

ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Processo su cui intervenire per il cambiamento di identità.
uid_t ***uid***	Nuova identità richiesta.

DESCRIZIONE

La funzione ***proc_sys_setuid()*** modifica l'identità del processo ***pid***, oppure tutti i tipi di identità, a seconda di certe condizioni:

- se l'identità efficace del processo ***pid*** è zero, viene modificata l'identità reale, quella salvata e quella efficace, utilizzando il nuovo valore ***uid***;

- se l'identità efficace del processo coincide già con quella del valore richiesto ***uid***, non viene apportata alcuna modifica e la funzione si conclude con successo;
- se l'identità reale o quella salvato del processo ***pid*** coincide con l'identità richiesta ***uid***, allora viene modificata l'identità efficace del processo con il valore ***uid***;
- in tutti gli altri casi si ottiene un errore.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_SETUID**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
-1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EPERM	Il processo ***pid*** non può cambiare identità come richiesto.

FILE SORGENTI

'lib/unistd/setuid.c' [i161.17.32]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_setuid.c' [i160.9.26]

VEDERE ANCHE

setuid(2) [u0.33], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_seteuid(9)* [i159.8.24].

os16: proc_sys_signal(9)

NOME

'**proc_sys_signal**' - modifica della configurazione dei segnali

SINTASSI

```
<kernel/proc.h>
sighandler_t proc_sys_signal (pid_t pid, int sig,
                              sighandler_t handler);
```

ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Processo su cui intervenire per il cambiamento di configurazione.
int ***sig***	Segnale da riconfigurare.
sighandler_t ***handler***	Nuova azione da associare al segnale. Si possono solo usare i valori corrispondenti a '**SIG_IGN**' e '**SIG_DFL**', con cui, rispettivamente, si inibisce il segnale o gli si attribuisce l'azione predefinita.

DESCRIZIONE

La funzione ***proc_sys_signal()*** ha il compito di modificare il comportamento del processo nel caso fosse ricevuto il segnale specificato. Teoricamente, il parametro ***handler*** potrebbe riferirsi a una funzione da eseguire allo scattare del segnale; tuttavia, os16 non è in grado di gestire questa evenienza e per ***handler*** si può specificare soltanto il valore corrispondente all'azione predefinita o a quella di inibizione del segnale.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_SIGNAL**'.

VALORE RESTITUITO

La funzione restituisce il valore di ***handler*** abbinato precedentemente al processo. Se si verifica un errore, restituisce '**SIG_ERR**' e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EINVAL	La combinazione degli argomenti non è valida.

FILE SORGENTI

'lib/signal/signal.c' [i161.8.2]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_signal.c' [i160.9.27]

VEDERE ANCHE

signal(2) [u0.34], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_kill(9)* [i159.8.23].

os16: proc_sys_wait(9)

NOME

'**proc_sys_wait**' - attesa per la morte di un processo figlio

SINTASSI

```
<kernel/proc.h>
pid_t proc_sys_wait (pid_t pid, int *status);
```

ARGOMENTI

Argomento	Descrizione
pid_t ***pid***	Il processo che intende mettersi in attesa della morte di un proprio figlio.
int ****status***	Puntatore a una variabile atta a contenere il valore di uscita di un processo figlio defunto.

DESCRIZIONE

La funzione ***proc_sys_wait()*** ha il compito di mettere il processo ***pid*** in pausa, fino alla morte di uno dei propri processi figli.

Per realizzare questo compito, la funzione scandisce inizialmente la tabella dei processi alla ricerca di figli di ***pid***. Se tra questi ne esiste già uno defunto, allora aggiorna ****status*** con il valore di uscita di quello, liberando definitivamente la tabella dei processi dalle tracce di questo figlio. Se invece, pur avendo trovato dei figli, questi risultano ancora tutti in funzione, mette il processo ***pid*** in pausa, in attesa di un segnale SIGCHLD.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_WAIT**'.

VALORE RESTITUITO

La funzione restituisce il numero PID del processo defunto, se c'è, aggiornando anche ****status*** con il valore di uscita dello stesso processo. Se invece il processo ***pid*** è stato messo in attesa, allora restituisce zero, mentre se non ci sono proprio figli di ***pid***, restituisce -1 e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
ECHILD	Non ci sono figli del processo ***pid*** e a nulla servirebbe attendere.

FILE SORGENTI

'lib/sys/wait/wait.c' [i161.15.1]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_wait.c' [i160.9.28]

VEDERE ANCHE

wait(2) [u0.43], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_available(9)* [i159.8.3], *proc_scheduler(9)* [i159.8.11], *proc_sys_fork(9)* [i159.8.22], *proc_sys_kill(9)* [i159.8.23].

os16: sysroutine(9)

NOME

'**sysroutine**' - attuazione delle chiamate di sistema

SINTASSI

```
<kernel/proc.h>
void sysroutine (uint16_t *sp, segment_t *segment_d,
                 uint16_t syscallnr,
                 uint16_t msg_off, uint16_t msg_size);
```

ARGOMENTI

Argomento	Descrizione
`uint16_t *sp`	Puntatore all'indice della pila dei dati del processo che ha emesso la chiamata di sistema.
`segment_t *segment_d`	Puntatore al valore del segmento dati del processo che ha emesso la chiamata di sistema.
`uint16_t syscallnr`	Il numero della chiamata di sistema.
`uint16_t msg_off`	Nonostante il tipo di variabile, si tratta del **puntatore** alla posizione di memoria in cui inizia il messaggio con gli argomenti della chiamata di sistema, ma tale puntatore è valido solo nell'ambito del segmento ***segment_d**.
`uint16_t msg_size`	La lunghezza del messaggio della chiamata di sistema.

DESCRIZIONE

La funzione ***sysroutine()*** viene chiamata esclusivamente dalla routine *isr_80(9)* [i159.8.1], a seguito di una chiamata di sistema.

Inizialmente, la funzione individua il processo elaborativo corrispondente a quello che utilizza il segmento dati ***segment_d** e l'**indirizzo efficace** dell'area di memoria contenente il messaggio della chiamata di sistema, traducendo le informazioni contenute in ***msg_off*** e ***segment_d**.

Attraverso un'unione di variabili strutturate, tutti i tipi di messaggi gestibili per le chiamate di sistema vengono dichiarati assieme in un'unica area di memoria. Successivamente, la funzione deve trasferire il messaggio, dall'indirizzo efficace calcolato precedentemente all'inizio dell'unione in questione.

Quando la funzione è in grado di accedere ai dati del messaggio, procede con una grande struttura di selezione, sulla base del tipo di messaggio, quindi esegue ciò che è richiesto, avvalendosi prevalentemente di altre funzioni, interpretando il messaggio in modo diverso a seconda del tipo di chiamata.

Il messaggio viene poi sovrascritto con le informazioni prodotte dall'azione richiesta, in particolare viene trasferito anche il valore della variabile ***errno*** del kernel, in modo che possa essere recepita anche dal processo che ha eseguito la chiamata, in caso di esito erroneo. Pertanto, il messaggio viene anche riscritto a partire dall'indirizzo efficace da cui era stato copiato precedentemente, in modo da renderlo disponibile effettivamente al processo chiamante.

Quando la funzione ***sysroutine()*** ha finito il suo lavoro, chiama a sua volta *proc_scheduler(9)* [i159.8.11], perché con l'occasione provveda eventualmente alla sostituzione del processo attivo con un altro che si trovi nello stato di pronto.

VALORE RESTITUITO

La funzione non restituisce alcun valore, in quanto tutto ciò che c'è da restituire viene trasmesso con la riscrittura del messaggio, nell'area di memoria originale.

FILE SORGENTI

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

VEDERE ANCHE

sys(2) [u0.37], *isr_80(9)* [i159.8.1], *proc_scheduler(9)* [i159.8.11], *dev_io(9)* [i159.1.1], *path_chdir(9)* [i159.3.30], *path_chmod(9)* [i159.3.31], *path_chown(9)* [i159.3.32], *fd_close(9)* [i159.3.3], *fd_dup(9)* [i159.3.4], *fd_dup2(9)* [i159.3.4], *proc_sys_exec(9)* [i159.8.20], *proc_sys_exit(9)* [i159.8.21], *fd_chmod(9)* [i159.3.1], *fd_chown(9)* [i159.3.2], *fd_fcntl(9)* [i159.3.6], *proc_sys_fork(9)* [i159.8.22], *fd_stat(9)* [i159.3.50], *proc_sys_kill(9)* [i159.8.23], *path_link(9)* [i159.3.38], *fd_lseek(9)* [i159.3.7], *path_mkdir(9)* [i159.3.39], *path_mknod(9)* [i159.3.40], *path_mount(9)* [i159.3.41], *fd_open(9)* [i159.3.8], *fd_read(9)* [i159.3.9], *proc_sys_seteuid(9)* [i159.8.24], *proc_sys_setuid(9)* [i159.8.25], *proc_sys_signal(9)* [i159.8.26], *path_stat(9)* [i159.3.50], *path_umount(9)* [i159.3.41], *path_unlink(9)* [i159.3.44], *proc_sys_wait(9)* [i159.8.27], *fd_write(9)* [i159.3.12].

os16: tty(9)

Il file 'kernel/tty.h' [u0.10] descrive le funzioni per la gestione dei terminali virtuali.

Per la descrizione dell'organizzazione della gestione dei terminali virtuali di os16, si rimanda alla sezione u146. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da quel capitolo.

Tabella u146.1. Funzioni per la gestione dei terminali, dichiarate nel file di intestazione 'kernel/tty.h'.

Funzione	Descrizione
`void tty_init (void);`	Inizializza la gestione dei terminali. Viene usata una volta sola nella funzione ***main()*** del kernel.
`tty_t *tty_reference (dev_t device);`	Restituisce il puntatore a un elemento della tabella dei terminali. Se come numero di dispositivo si indica lo zero, si ottiene il riferimento a tutta la tabella; se non viene trovato il numero di dispositivo cercato, si ottiene il puntatore nullo.
`dev_t tty_console (dev_t device);`	Seleziona la console indicata attraverso il numero di dispositivo che costituisce l'unico parametro. Se viene dato un valore a zero, si ottiene solo di conoscere qual è la console attiva. La console selezionata viene anche memorizzata in una variabile statica, per le chiamate successive della funzione. Se viene indicato un numero di dispositivo non valido, si seleziona implicitamente la prima console.

Funzione	Descrizione
`int tty_read (dev_t device);`	Legge un carattere dal terminale specificato attraverso il numero di dispositivo. Per la precisione, il carattere viene tratto dal campo relativo contenuto nella tabella dei terminali. Il carattere viene restituito dalla funzione come valore intero comune; se si ottiene zero significa che non è disponibile alcun carattere.
`void tty_write (dev_t device, int c);`	Scrive sullo schermo del terminale rappresentato dal numero di dispositivo, il carattere fornito come secondo parametro.

Parte xxxvi

Codice di os16

Script e sorgenti del kernel 1589

os16: directory principale 1597
os16: «kernel/devices.h» 1602
os16: «kernel/diag.h» 1606
os16: «kernel/fs.h» 1616
os16: «kernel/ibm_i86.h» 1669
os16: «kernel/k_libc.h» 1682
os16: «kernel/main.h» 1685
os16: «kernel/memory.h» 1690
os16: «kernel/proc.h» 1694
os16: «kernel/tty.h» 1723

Sorgenti della libreria generale 1727

os16: file isolati della directory «lib/» 1735
os16: «lib/dirent.h» 1741
os16: «lib/errno.h» 1744
os16: «lib/fcntl.h» 1747
os16: «lib/grp.h» 1749
os16: «lib/libgen.h» 1749
os16: «lib/pwd.h» 1751
os16: «lib/signal.h» 1752
os16: «lib/stdio.h» 1753
os16: «lib/stdlib.h» 1788
os16: «lib/string.h» 1802
os16: «lib/sys/os16.h» 1811
os16: «lib/sys/stat.h» 1821
os16: «lib/sys/types.h» 1825
os16: «lib/sys/wait.h» 1826
os16: «lib/time.h» 1826
os16: «lib/unistd.h» 1831
os16: «lib/utime.h» 1847

Sorgenti delle applicazioni 1849

os16: directory «applic/» 1850

os16: directory principale 1597
bochs 1597
qemu 1597
makeit 1597
os16: «kernel/devices.h» 1602
kernel/devices/dev_dsk.c 1602
kernel/devices/dev_io.c 1602
kernel/devices/dev_kmem.c 1603
kernel/devices/dev_mem.c 1604
kernel/devices/dev_tty.c 1605
os16: «kernel/diag.h» 1606
kernel/diag/print_fd.c 1607
kernel/diag/print_fd_head.c 1607
kernel/diag/print_fd_list.c 1608
kernel/diag/print_file_head.c 1608
kernel/diag/print_file_list.c 1608
kernel/diag/print_file_num.c 1608
kernel/diag/print_hex_16.c 1609
kernel/diag/print_hex_16_reverse.c 1609
kernel/diag/print_hex_32.c 1609
kernel/diag/print_hex_32_reverse.c 1609
kernel/diag/print_hex_8.c 1609
kernel/diag/print_hex_8_reverse.c 1610
kernel/diag/print_inode.c 1610
kernel/diag/print_inode_head.c 1610
kernel/diag/print_inode_list.c 1611
kernel/diag/print_inode_map.c 1611
kernel/diag/print_inode_zone_list.c 1611
kernel/diag/print_inode_zones.c 1611
kernel/diag/print_inode_zones_head.c 1612
kernel/diag/print_kmem.c 1612
kernel/diag/print_mb_map.c 1612
kernel/diag/print_memory_map.c 1612
kernel/diag/print_proc_head.c 1613
kernel/diag/print_proc_list.c 1613
kernel/diag/print_proc_pid.c 1613
kernel/diag/print_segments.c 1614
kernel/diag/print_superblock.c 1614
kernel/diag/print_time.c 1614
kernel/diag/print_zone_map.c 1614
kernel/diag/reverse_16_bit.c 1615
kernel/diag/reverse_32_bit.c 1615
kernel/diag/reverse_8_bit.c 1615
os16: «kernel/fs.h» 1616
kernel/fs/fd_chmod.c 1618
kernel/fs/fd_chown.c 1619
kernel/fs/fd_close.c 1619
kernel/fs/fd_dup.c 1620
kernel/fs/fd_dup2.c 1620
kernel/fs/fd_fcntl.c 1621
kernel/fs/fd_lseek.c 1622
kernel/fs/fd_open.c 1623
kernel/fs/fd_read.c 1625
kernel/fs/fd_reference.c 1626
kernel/fs/fd_stat.c 1626

kernel/fs/fd_write.c 1627
kernel/fs/file_reference.c 1628
kernel/fs/file_stdio_dev_make.c 1629
kernel/fs/file_table.c 1629
kernel/fs/inode_alloc.c 1629
kernel/fs/inode_check.c 1631
kernel/fs/inode_dir_empty.c 1632
kernel/fs/inode_file_read.c 1632
kernel/fs/inode_file_write.c 1633
kernel/fs/inode_free.c 1635
kernel/fs/inode_fzones_read.c 1635
kernel/fs/inode_fzones_write.c 1636
kernel/fs/inode_get.c 1636
kernel/fs/inode_put.c 1638
kernel/fs/inode_reference.c 1639
kernel/fs/inode_save.c 1640
kernel/fs/inode_stdio_dev_make.c 1641
kernel/fs/inode_table.c 1641
kernel/fs/inode_truncate.c 1641
kernel/fs/inode_zone.c 1643
kernel/fs/path_chdir.c 1647
kernel/fs/path_chmod.c 1647
kernel/fs/path_chown.c 1648
kernel/fs/path_device.c 1649
kernel/fs/path_fix.c 1649
kernel/fs/path_full.c 1650
kernel/fs/path_inode.c 1650
kernel/fs/path_inode_link.c 1653
kernel/fs/path_link.c 1655
kernel/fs/path_mkdir.c 1656
kernel/fs/path_mknod.c 1657
kernel/fs/path_mount.c 1658
kernel/fs/path_stat.c 1659
kernel/fs/path_umount.c 1659
kernel/fs/path_unlink.c 1661
kernel/fs/sb_inode_status.c 1663
kernel/fs/sb_mount.c 1663
kernel/fs/sb_reference.c 1665
kernel/fs/sb_save.c 1665
kernel/fs/sb_table.c 1666
kernel/fs/sb_zone_status.c 1666
kernel/fs/zone_alloc.c 1666
kernel/fs/zone_free.c 1667
kernel/fs/zone_read.c 1668
kernel/fs/zone_write.c 1668
os16: «kernel/ibm_i86.h» 1669
kernel/ibm_i86/_cli.s 1670
kernel/ibm_i86/_in_16.s 1670
kernel/ibm_i86/_in_8.s 1670
kernel/ibm_i86/_int10_00.s 1671
kernel/ibm_i86/_int10_02.s 1671
kernel/ibm_i86/_int10_05.s 1671
kernel/ibm_i86/_int12.s 1672
kernel/ibm_i86/_int13_00.s 1672
kernel/ibm_i86/_int13_02.s 1672
kernel/ibm_i86/_int13_03.s 1673
kernel/ibm_i86/_int16_00.s 1673
kernel/ibm_i86/_int16_01.s 1674

kernel/ibm_i86/_int16_02.s 1674
kernel/ibm_i86/_out_16.s 1675
kernel/ibm_i86/_out_8.s 1675
kernel/ibm_i86/_ram_copy.s 1675
kernel/ibm_i86/_sti.s 1675
kernel/ibm_i86/con_char_read.c 1676
kernel/ibm_i86/con_char_ready.c 1676
kernel/ibm_i86/con_char_wait.c 1676
kernel/ibm_i86/con_init.c 1677
kernel/ibm_i86/con_putc.c 1677
kernel/ibm_i86/con_scroll.c 1678
kernel/ibm_i86/con_select.c 1678
kernel/ibm_i86/dsk_read_bytes.c 1678
kernel/ibm_i86/dsk_read_sectors.c 1679
kernel/ibm_i86/dsk_reset.c 1680
kernel/ibm_i86/dsk_sector_to_chs.c 1680
kernel/ibm_i86/dsk_setup.c 1680
kernel/ibm_i86/dsk_table.c 1680
kernel/ibm_i86/dsk_write_bytes.c 1680
kernel/ibm_i86/dsk_write_sectors.c 1681
kernel/ibm_i86/irq_off.c 1682
kernel/ibm_i86/irq_on.c 1682
os16: «kernel/k_libc.h» 1682
kernel/k_libc/k_clock.c 1683
kernel/k_libc/k_close.c 1683
kernel/k_libc/k_exit.s 1683
kernel/k_libc/k_kill.c 1683
kernel/k_libc/k_open.c 1683
kernel/k_libc/k_perror.c 1684
kernel/k_libc/k_printf.c 1684
kernel/k_libc/k_puts.c 1684
kernel/k_libc/k_read.c 1684
kernel/k_libc/k_stime.c 1685
kernel/k_libc/k_time.c 1685
kernel/k_libc/k_vprintf.c 1685
kernel/k_libc/k_vsprintf.c 1685
os16: «kernel/main.h» 1685
kernel/main/build.h 1685
kernel/main/crt0.s 1686
kernel/main/main.c 1688
kernel/main/menu.c 1689
kernel/main/run.c 1690
os16: «kernel/memory.h» 1690
kernel/memory/address.c 1690
kernel/memory/mb_alloc.c 1691
kernel/memory/mb_alloc_size.c 1691
kernel/memory/mb_free.c 1692
kernel/memory/mb_reference.c 1693
kernel/memory/mb_table.c 1693
kernel/memory/mem_copy.c 1693
kernel/memory/mem_read.c 1694
kernel/memory/mem_write.c 1694
os16: «kernel/proc.h» 1694
kernel/proc/_isr.s 1695
kernel/proc/_ivt_load.s 1698
kernel/proc/proc_available.c 1698
kernel/proc/proc_dump_memory.c 1699

kernel/proc/proc_find.c 1699
kernel/proc/proc_init.c 1700
kernel/proc/proc_reference.c 1701
kernel/proc/proc_sch_signals.c 1701
kernel/proc/proc_sch_terminals.c 1702
kernel/proc/proc_sch_timers.c 1703
kernel/proc/proc_scheduler.c 1703
kernel/proc/proc_sig_chld.c 1705
kernel/proc/proc_sig_cont.c 1705
kernel/proc/proc_sig_core.c 1705
kernel/proc/proc_sig_ignore.c 1706
kernel/proc/proc_sig_off.c 1706
kernel/proc/proc_sig_on.c 1706
kernel/proc/proc_sig_status.c 1706
kernel/proc/proc_sig_stop.c 1707
kernel/proc/proc_sig_term.c 1707
kernel/proc/proc_sys_exec.c 1707
kernel/proc/proc_sys_exit.c 1713
kernel/proc/proc_sys_fork.c 1715
kernel/proc/proc_sys_kill.c 1717
kernel/proc/proc_sys_seteuid.c 1718
kernel/proc/proc_sys_setuid.c 1718
kernel/proc/proc_sys_signal.c 1719
kernel/proc/proc_sys_wait.c 1719
kernel/proc/proc_table.c 1720
kernel/proc/sysroutine.c 1720
os16: «kernel/tty.h» 1723
kernel/tty/tty_console.c 1724
kernel/tty/tty_init.c 1724
kernel/tty/tty_read.c 1725
kernel/tty/tty_reference.c 1725
kernel/tty/tty_table.c 1725
kernel/tty/tty_write.c 1725

address.c 1690 bochs 1597 build.h 1685
con_char_read.c 1676 con_char_ready.c 1676
con_char_wait.c 1676 con_init.c 1677 con_putc.c 1677 con_scroll.c 1678 con_select.c 1678 crt0.s 1686 devices.h 1602 dev_dsk.c 1602 dev_io.c 1602
dev_kmem.c 1603 dev_mem.c 1604 dev_tty.c 1605
diag.h 1606 dsk_read_bytes.c 1678
dsk_read_sectors.c 1679 dsk_reset.c 1680
dsk_sector_to_chs.c 1680 dsk_setup.c 1680
dsk_table.c 1680 dsk_write_bytes.c 1680
dsk_write_sectors.c 1681 fd_chmod.c 1618
fd_chown.c 1619 fd_close.c 1619 fd_dup.c 1620
fd_dup2.c 1620 fd_fcntl.c 1621 fd_lseek.c 1622
fd_open.c 1623 fd_read.c 1625 fd_reference.c 1626
fd_stat.c 1626 fd_write.c 1627 file_reference.c 1628 file_stdio_dev_make.c 1629 file_table.c 1629
fs.h 1616 ibm_i86.h 1669 inode_alloc.c 1629
inode_check.c 1631 inode_dir_empty.c 1632
inode_file_read.c 1632 inode_file_write.c 1633
inode_free.c 1635 inode_fzones_read.c 1635
inode_fzones_write.c 1636 inode_get.c 1636
inode_put.c 1638 inode_reference.c 1639
inode_save.c 1640 inode_stdio_dev_make.c 1641
inode_table.c 1641 inode_truncate.c 1641
inode_zone.c 1643 irq_off.c 1682 irq_on.c 1682
k_clock.c 1683 k_close.c 1683 k_exit.s 1683
k_kill.c 1683 k_libc.h 1682 k_open.c 1683
k_perror.c 1684 k_printf.c 1684 k_puts.c 1684

k_read.c 1684 k_stime.c 1685 k_time.c 1685
k_vprintf.c 1685 k_vsprintf.c 1685 main.c 1688
main.h 1685 makeit 1597 mb_alloc.c 1691
mb_alloc_size.c 1691 mb_free.c 1692
mb_reference.c 1693 mb_table.c 1693 memory.h 1690
mem_copy.c 1693 mem_read.c 1694 mem_write.c 1694
menu.c 1689 path_chdir.c 1647 path_chmod.c 1647
path_chown.c 1648 path_device.c 1649 path_fix.c 1649 path_full.c 1650 path_inode.c 1650
path_inode_link.c 1653 path_link.c 1655
path_mkdir.c 1656 path_mknod.c 1657 path_mount.c 1658 path_stat.c 1659 path_umount.c 1659
path_unlink.c 1661 print_fd.c 1607
print_fd_head.c 1607 print_fd_list.c 1608
print_file_head.c 1608 print_file_list.c 1608
print_file_num.c 1608 print_hex_16.c 1609
print_hex_16_reverse.c 1609 print_hex_32.c 1609
print_hex_32_reverse.c 1609 print_hex_8.c 1609
print_hex_8_reverse.c 1610 print_inode.c 1610
print_inode_head.c 1610 print_inode_list.c 1611
print_inode_map.c 1611 print_inode_zones.c 1611 print_inode_zones_head.c 1612
print_inode_zone_list.c 1611 print_kmem.c 1612
print_mb_map.c 1612 print_memory_map.c 1612
print_proc_head.c 1613 print_proc_list.c 1613
print_proc_pid.c 1613 print_segments.c 1614
print_superblock.c 1614 print_time.c 1614
print_zone_map.c 1614 proc.h 1694
proc_available.c 1698 proc_dump_memory.c 1699
proc_find.c 1699 proc_init.c 1700
proc_reference.c 1701 proc_scheduler.c 1703
proc_sch_signals.c 1701 proc_sch_terminals.c 1702 proc_sch_timers.c 1703 proc_sig_chld.c 1705
proc_sig_cont.c 1705 proc_sig_core.c 1705
proc_sig_ignore.c 1706 proc_sig_off.c 1706
proc_sig_on.c 1706 proc_sig_status.c 1706
proc_sig_stop.c 1707 proc_sig_term.c 1707
proc_sys_exec.c 1707 proc_sys_exit.c 1713
proc_sys_fork.c 1715 proc_sys_kill.c 1717
proc_sys_seteuid.c 1718 proc_sys_setuid.c 1718
proc_sys_signal.c 1719 proc_sys_wait.c 1719
proc_table.c 1720 qemu 1597 reverse_16_bit.c 1615 reverse_32_bit.c 1615 reverse_8_bit.c 1615 run.c 1690 sb_inode_status.c 1663 sb_mount.c 1663
sb_reference.c 1665 sb_save.c 1665 sb_table.c 1666
sb_zone_status.c 1666 sysroutine.c 1720 tty.h 1723
tty_console.c 1724 tty_init.c 1724 tty_read.c 1725
tty_reference.c 1725 tty_table.c 1725
tty_write.c 1725 zone_alloc.c 1666 zone_free.c 1667 zone_read.c 1668 zone_write.c 1668 _cli.s 1670
_int10_00.s 1671 _int10_02.s 1671 _int10_05.s 1671
_int12.s 1672 _int13_00.s 1672 _int13_02.s 1672
_int13_03.s 1673 _int16_00.s 1673 _int16_01.s 1674
_int16_02.s 1674 _in_16.s 1670 _in_8.s 1670 _isr.s 1695 _ivt_load.s 1698 _out_16.s 1675 _out_8.s 1675
_ram_copy.s 1675 _sti.s 1675
os16: directory principale 1597
bochs 1597
qemu 1597
makeit 1597
os16: «kernel/devices.h» 1602
kernel/devices/dev_dsk.c 1602
kernel/devices/dev_io.c 1602
kernel/devices/dev_kmem.c 1603
kernel/devices/dev_mem.c 1604

kernel/devices/dev_tty.c 1605
os16: «kernel/diag.h» 1606
kernel/diag/print_fd.c 1607
kernel/diag/print_fd_head.c 1607
kernel/diag/print_fd_list.c 1608
kernel/diag/print_file_head.c 1608
kernel/diag/print_file_list.c 1608
kernel/diag/print_file_num.c 1608
kernel/diag/print_hex_16.c 1609
kernel/diag/print_hex_16_reverse.c 1609
kernel/diag/print_hex_32.c 1609
kernel/diag/print_hex_32_reverse.c 1609
kernel/diag/print_hex_8.c 1609
kernel/diag/print_hex_8_reverse.c 1610
kernel/diag/print_inode.c 1610
kernel/diag/print_inode_head.c 1610
kernel/diag/print_inode_list.c 1611
kernel/diag/print_inode_map.c 1611
kernel/diag/print_inode_zone_list.c 1611
kernel/diag/print_inode_zones.c 1611
kernel/diag/print_inode_zones_head.c 1612
kernel/diag/print_kmem.c 1612
kernel/diag/print_mb_map.c 1612
kernel/diag/print_memory_map.c 1612
kernel/diag/print_proc_head.c 1613
kernel/diag/print_proc_list.c 1613
kernel/diag/print_proc_pid.c 1613
kernel/diag/print_segments.c 1614
kernel/diag/print_superblock.c 1614
kernel/diag/print_time.c 1614
kernel/diag/print_zone_map.c 1614
kernel/diag/reverse_16_bit.c 1615
kernel/diag/reverse_32_bit.c 1615
kernel/diag/reverse_8_bit.c 1615
os16: «kernel/fs.h» 1616
kernel/fs/fd_chmod.c 1618
kernel/fs/fd_chown.c 1619
kernel/fs/fd_close.c 1619
kernel/fs/fd_dup.c 1620
kernel/fs/fd_dup2.c 1620
kernel/fs/fd_fcntl.c 1621
kernel/fs/fd_lseek.c 1622
kernel/fs/fd_open.c 1623
kernel/fs/fd_read.c 1625
kernel/fs/fd_reference.c 1626
kernel/fs/fd_stat.c 1626
kernel/fs/fd_write.c 1627
kernel/fs/file_reference.c 1628
kernel/fs/file_stdio_dev_make.c 1629
kernel/fs/file_table.c 1629
kernel/fs/inode_alloc.c 1629
kernel/fs/inode_check.c 1631
kernel/fs/inode_dir_empty.c 1632
kernel/fs/inode_file_read.c 1632
kernel/fs/inode_file_write.c 1633
kernel/fs/inode_free.c 1635
kernel/fs/inode_fzones_read.c 1635

kernel/fs/inode_fzones_write.c 1636
kernel/fs/inode_get.c 1636
kernel/fs/inode_put.c 1638
kernel/fs/inode_reference.c 1639
kernel/fs/inode_save.c 1640
kernel/fs/inode_stdio_dev_make.c 1641
kernel/fs/inode_table.c 1641
kernel/fs/inode_truncate.c 1641
kernel/fs/inode_zone.c 1643
kernel/fs/path_chdir.c 1647
kernel/fs/path_chmod.c 1647
kernel/fs/path_chown.c 1648
kernel/fs/path_device.c 1649
kernel/fs/path_fix.c 1649
kernel/fs/path_full.c 1650
kernel/fs/path_inode.c 1650
kernel/fs/path_inode_link.c 1653
kernel/fs/path_link.c 1655
kernel/fs/path_mkdir.c 1656
kernel/fs/path_mknod.c 1657
kernel/fs/path_mount.c 1658
kernel/fs/path_stat.c 1659
kernel/fs/path_umount.c 1659
kernel/fs/path_unlink.c 1661
kernel/fs/sb_inode_status.c 1663
kernel/fs/sb_mount.c 1663
kernel/fs/sb_reference.c 1665
kernel/fs/sb_save.c 1665
kernel/fs/sb_table.c 1666
kernel/fs/sb_zone_status.c 1666
kernel/fs/zone_alloc.c 1666
kernel/fs/zone_free.c 1667
kernel/fs/zone_read.c 1668
kernel/fs/zone_write.c 1668
os16: «kernel/ibm_i86.h» 1669
kernel/ibm_i86/_cli.s 1670
kernel/ibm_i86/_in_16.s 1670
kernel/ibm_i86/_in_8.s 1670
kernel/ibm_i86/_int10_00.s 1671
kernel/ibm_i86/_int10_02.s 1671
kernel/ibm_i86/_int10_05.s 1671
kernel/ibm_i86/_int12.s 1672
kernel/ibm_i86/_int13_00.s 1672
kernel/ibm_i86/_int13_02.s 1672
kernel/ibm_i86/_int13_03.s 1673
kernel/ibm_i86/_int16_00.s 1673
kernel/ibm_i86/_int16_01.s 1674
kernel/ibm_i86/_int16_02.s 1674
kernel/ibm_i86/_out_16.s 1675
kernel/ibm_i86/_out_8.s 1675
kernel/ibm_i86/_ram_copy.s 1675
kernel/ibm_i86/_sti.s 1675
kernel/ibm_i86/con_char_read.c 1676
kernel/ibm_i86/con_char_ready.c 1676
kernel/ibm_i86/con_char_wait.c 1676
kernel/ibm_i86/con_init.c 1677
kernel/ibm_i86/con_putc.c 1677
kernel/ibm_i86/con_scroll.c 1678

kernel/ibm_i86/con_select.c 1678
kernel/ibm_i86/dsk_read_bytes.c 1678
kernel/ibm_i86/dsk_read_sectors.c 1679
kernel/ibm_i86/dsk_reset.c 1680
kernel/ibm_i86/dsk_sector_to_chs.c 1680
kernel/ibm_i86/dsk_setup.c 1680
kernel/ibm_i86/dsk_table.c 1680
kernel/ibm_i86/dsk_write_bytes.c 1680
kernel/ibm_i86/dsk_write_sectors.c 1681
kernel/ibm_i86/irq_off.c 1682
kernel/ibm_i86/irq_on.c 1682
os16: «kernel/k_libc.h» 1682
kernel/k_libc/k_clock.c 1683
kernel/k_libc/k_close.c 1683
kernel/k_libc/k_exit.s 1683
kernel/k_libc/k_kill.c 1683
kernel/k_libc/k_open.c 1683
kernel/k_libc/k_perror.c 1684
kernel/k_libc/k_printf.c 1684
kernel/k_libc/k_puts.c 1684
kernel/k_libc/k_read.c 1684
kernel/k_libc/k_stime.c 1685
kernel/k_libc/k_time.c 1685
kernel/k_libc/k_vprintf.c 1685
kernel/k_libc/k_vsprintf.c 1685
os16: «kernel/main.h» 1685
kernel/main/build.h 1685
kernel/main/crt0.s 1686
kernel/main/main.c 1688
kernel/main/menu.c 1689
kernel/main/run.c 1690
os16: «kernel/memory.h» 1690
kernel/memory/address.c 1690
kernel/memory/mb_alloc.c 1691
kernel/memory/mb_alloc_size.c 1691
kernel/memory/mb_free.c 1692
kernel/memory/mb_reference.c 1693
kernel/memory/mb_table.c 1693
kernel/memory/mem_copy.c 1693
kernel/memory/mem_read.c 1694
kernel/memory/mem_write.c 1694
os16: «kernel/proc.h» 1694
kernel/proc/_isr.s 1695
kernel/proc/_ivt_load.s 1698
kernel/proc/proc_available.c 1698
kernel/proc/proc_dump_memory.c 1699
kernel/proc/proc_find.c 1699
kernel/proc/proc_init.c 1700
kernel/proc/proc_reference.c 1701
kernel/proc/proc_sch_signals.c 1701
kernel/proc/proc_sch_terminals.c 1702
kernel/proc/proc_sch_timers.c 1703
kernel/proc/proc_scheduler.c 1703
kernel/proc/proc_sig_chld.c 1705
kernel/proc/proc_sig_cont.c 1705
kernel/proc/proc_sig_core.c 1705
kernel/proc/proc_sig_ignore.c 1706

kernel/proc/proc_sig_off.c 1706
kernel/proc/proc_sig_on.c 1706
kernel/proc/proc_sig_status.c 1706
kernel/proc/proc_sig_stop.c 1707
kernel/proc/proc_sig_term.c 1707
kernel/proc/proc_sys_exec.c 1707
kernel/proc/proc_sys_exit.c 1713
kernel/proc/proc_sys_fork.c 1715
kernel/proc/proc_sys_kill.c 1717
kernel/proc/proc_sys_seteuid.c 1718
kernel/proc/proc_sys_setuid.c 1718
kernel/proc/proc_sys_signal.c 1719
kernel/proc/proc_sys_wait.c 1719
kernel/proc/proc_table.c 1720
kernel/proc/sysroutine.c 1720
os16: «kernel/tty.h» 1723
kernel/tty/tty_console.c 1724
kernel/tty/tty_init.c 1724
kernel/tty/tty_read.c 1725
kernel/tty/tty_reference.c 1725
kernel/tty/tty_table.c 1725
kernel/tty/tty_write.c 1725

os16: directory principale

«

bochs

«

Si veda la sezione u0.2.

```
10001  #!/bin/sh
10002
10003  bochs -q "boot:floppy" \
10004           "floppya: 1_44=floppy.a, status=inserted" \
10005           "floppyb: 1_44=floppy.b, status=inserted" \
10006           "keyboard_mapping: enabled=1, \
10007            map=/usr/share/bochs/keymaps/x11-pc-it.map" \
10008           "keyboard_type: xt" \
10009           "vga: none" \
10010           "romimage: file=\"/usr/share/bochs/BIOS-bochs-legacy\"" \
10011           "megs:1"
```

qemu

«

Si veda la sezione u0.2.

```
20001  #!/bin/sh
20002
20003  qemu -fda floppy.a \
20004       -fdb floppy.b \
20005       -boot order=a
20006
```

makeit

«

Si veda la sezione u0.2.

```
30001  #!/bin/sh
30002  #
30003  # makeit...
30004  #
30005  OPTION="$1"
30006  OS16PATH=""
30007  #
30008  edition () {
30009      local EDITION="kernel/main/build.h"
30010      echo -n                                > $EDITION
30011      echo -n "#define BUILD_DATE \""      >> $EDITION
30012      echo -n `date "+%Y.%m.%d %H:%M:%S"` >> $EDITION
30013      echo  "\""                           >> $EDITION
30014  }
30015  #
30016  #
30017  #
30018  makefile () {
30019      #
30020      local MAKEFILE="Makefile"
30021      local TAB="	"
30022      #
30023      local SOURCE_C=""
30024      local C=""
30025      local SOURCE_S=""
```

```
30026     local S=""
30027     #
30028     local c
30029     local s
30030     #
30031     # Trova i file in C.
30032     #
30033     for c in *.c
30034     do
30035         if [ -f $c ]
30036         then
30037         C=‘basename $c .c‘
30038             SOURCE_C="$SOURCE_C $C"
30039         fi
30040     done
30041     #
30042     # Trova i file in ASM.
30043     #
30044     for s in *.s
30045     do
30046         if [ -f $s ]
30047         then
30048         S=‘basename $s .s‘
30049             SOURCE_S="$SOURCE_S $S"
30050         fi
30051     done
30052     #
30053     # Prepara il file make.
30054     # GCC viene usato per potenziare il controllo degli errori.
30055     #
30056     echo -n                                                  > $MAKEFILE
30057     echo "# This file was made automatically"               >> $MAKEFILE
30058     echo "# by the script \‘makeit\’, based on the"         >> $MAKEFILE
30059     echo "# directory content."                             >> $MAKEFILE
30060     echo "# Please use \‘makeit\’ to compile and"          >> $MAKEFILE
30061     echo "# \‘makeit clean\’ to clean directories."         >> $MAKEFILE
30062     echo "#"                                              >> $MAKEFILE
30063     echo "c = $SOURCE_C"                                  >> $MAKEFILE
30064     echo "#"                                              >> $MAKEFILE
30065     echo "s = $SOURCE_S"                                  >> $MAKEFILE
30066     echo "#"                                              >> $MAKEFILE
30067     echo "all: \$(s) \$(c)"                               >> $MAKEFILE
30068     echo "#"                                              >> $MAKEFILE
30069     echo "clean:"                                         >> $MAKEFILE
30070     echo "${TAB}@rm \$(c) \$(s) *.o *.assembler 2> /dev/null ; true" \
30071                                                           >> $MAKEFILE
30072     echo "${TAB}@rm *.symbols 2> /dev/null     ; true"    >> $MAKEFILE
30073     echo "${TAB}@pwd"                                     >> $MAKEFILE
30074     echo "#"                                              >> $MAKEFILE
30075     echo "\$(c):"                                         >> $MAKEFILE
30076     echo "${TAB}@echo \$@.c"                              >> $MAKEFILE
30077     echo "${TAB}@gcc -Wall -c -o \$@.o " \
30078                         "-I " \
30079                         "-I. " \
30080                         "-I$OS16PATH/lib " \
30081                         "-I$OS16PATH/ " \
30082                         "\$@.c"                           >> $MAKEFILE
30083     echo "${TAB}@rm \$@.o"                                >> $MAKEFILE
30084     echo "${TAB}@bcc -ansi -0 -Mc -S -o \$@.assembler " \
30085                         "-I " \
30086                         "-I. " \
30087                         "-I$OS16PATH/lib " \
30088                         "-I$OS16PATH/ " \
30089                         "\$@.c"                           >> $MAKEFILE
30090     echo "${TAB}@bcc -ansi -0 -Mc -c -o \$@.o " \
30091                         "-I " \
30092                         "-I. " \
30093                         "-I$OS16PATH/lib " \
30094                         "-I$OS16PATH/ " \
30095                         "\$@.c"                           >> $MAKEFILE
30096     echo "#"                                              >> $MAKEFILE
30097     echo "\$(s):"                                         >> $MAKEFILE
30098     echo "${TAB}@echo \$@.s"                              >> $MAKEFILE
30099     echo "${TAB}@as86 -u -0 -o \$@.o -s \$@.symbols \$@.s"  >> $MAKEFILE
30100     #
30101 }
30102 #
30103 #
30104 #
30105 main () {
30106     #
30107     local CURDIR=‘pwd‘
30108     local OBJECTS
30109     local OBJLIB
30110     local EXEC
30111     local BASENAME
30112     local PROGNAME
30113     local d
30114     local c
30115     local s
30116     local o
30117     #
30118     edition
30119     #
30120     # Copia dello scheletro
30121     #
30122     if [ "$OPTION" = "clean" ]
30123     then
30124         #
30125         # La copia non va fatta.
30126         #
```



```
30127         true
30128     else
30129         cp -dpRv skel/etc   /mnt/os16.a/
30130         cp -dpRv skel/dev   /mnt/os16.a/
30131         mkdir               /mnt/os16.a/mnt/
30132         mkdir               /mnt/os16.a/tmp/
30133         chmod 0777          /mnt/os16.a/tmp/
30134         mkdir               /mnt/os16.a/usr/
30135         cp -dpRv skel/root  /mnt/os16.a/
30136         cp -dpRv skel/home  /mnt/os16.a/
30137         cp -dpRv skel/usr/* /mnt/os16.b/
30138     fi
30139     #
30140     #
30141     #
30142     for d in ‘find kernel‘  \
30143              ‘find lib‘     \
30144              ‘find applic‘  \
30145              ‘find ported‘
30146     do
30147         if [ -d "$d" ]
30148         then
30149             #
30150             # Sono presenti dei file C o ASM?
30151             #
30152             c=‘echo $d/*.c | sed "s/ .*//"‘
30153             s=‘echo $d/*.s | sed "s/ .*//"‘
30154             #
30155             if [ -f "$c" ] || [ -f "$s" ]
30156             then
30157                 #
30158                 # Sì
30159                 #
30160                 CURDIR=‘pwd‘
30161                 cd $d
30162                 #
30163                 # Ricrea il file make
30164                 #
30165                 makefile
30166                 #
30167                 # Pulisce quindi la directory
30168                 #
30169                 make clean
30170                 #
30171                 #
30172                 #
30173                 if [ "$OPTION" = "clean" ]
30174                 then
30175                     #
30176                     # È stata richiesta la pulitura, ma questa
30177                     # è appena stata fatta!
30178                     #
30179                     true
30180                 else
30181                     #
30182                     # Qualunque altro argomento viene considerato
30183                     # un ‘make‘.
30184                     #
30185                     if ! make
30186                     then
30187                         #
30188                         # La compilazione è fallita.
30189                         #
30190                         cd "$CURDIR"
30191                         exit
30192                     fi
30193                 fi
30194                 cd "$CURDIR"
30195             fi
30196         fi
30197     done
30198     #
30199     cd "$CURDIR"
30200     #
30201     # Link
30202     #
30203     if [ "$OPTION" = "clean" ]
30204     then
30205         #
30206         # Il collegamento non va fatto.
30207         #
30208         true
30209     else
30210         #
30211         # Collegamento dei file del kernel.
30212         #
30213         OBJECTS=""
30214         #
30215         for o in ‘find kernel   -name \*.o -print‘ \
30216                  ‘find lib      -name \*.o -print‘
30217         do
30218             if    [ "$o" = "./kernel/main/crt0.o" ] \
30219                || [ "$o" = "./kernel/main/main.o" ] \
30220                || [ ! -e "$o" ]
30221             then
30222                 true
30223             else
30224                 OBJECTS="$OBJECTS $o"
30225             fi
30226         done
30227         #
```

```
30228         echo "Link"
30229         #
30230         ld86 -i -d -s -m -o kimage    \
30231                 kernel/main/crt0.o    \
30232                 kernel/main/main.o    \
30233                 $OBJECTS
30234         #
30235         # Copia il kernel nel dischetto.
30236         #
30237         if mount | grep /mnt/os16.a > /dev/null
30238         then
30239             cp -f kimage /mnt/os16.a/boot
30240         else
30241             echo "[$0] Cannot copy the kernel image "
30242             echo "[$0]   inside the floppy disk image!"
30243         fi
30244         sync
30245         #
30246         # Collegamento delle applicazioni di os16.
30247         #
30248         OBJLIB=""
30249         #
30250         for o in `find lib      -name \*.o -print`
30251         do
30252             OBJLIB="$OBJLIB $o"
30253         done
30254         #
30255         # Scansione delle applicazioni interne.
30256         #
30257         for o in `find applic     -name \*.o -print`
30258         do
30259             if    [ "$o" = "applic/crt0.o" ] \
30260                || [ ! -e "$o" ] \
30261                || echo "$o" | grep ".crt0.o$" > /dev/null
30262             then
30263                 #
30264                 # Il file non esiste oppure si tratta di `...crt0.s'.
30265                 #
30266                 true
30267             else
30268                 #
30269                 # File oggetto differente da `...crt0.s'.
30270                 #
30271                 EXEC=`echo "$o" | sed "s/\.o$//"`
30272                 BASENAME=`basename $o .o`
30273                 if [ -e "applic/$BASENAME.crt0.o" ]
30274                 then
30275                     #
30276                     # Qui c'è un file `...crt0.o' specifico.
30277                     #
30278                     ld86 -i -d -s -o $EXEC \
30279                          applic/$BASENAME.crt0.o $o $OBJLIB
30280                 else
30281                     #
30282                     # Qui si usa il file `crt0.o' generale.
30283                     #
30284                     ld86 -i -d -s -o $EXEC applic/crt0.o $o $OBJLIB
30285                 fi
30286                 #
30287                 if [ -x "applic/$BASENAME" ]
30288                 then
30289                     if mount | grep /mnt/os16.a > /dev/null
30290                     then
30291                         mkdir /mnt/os16.a/bin/ 2> /dev/null
30292                         cp -f "$EXEC" /mnt/os16.a/bin
30293                     else
30294                         echo "[$0] Cannot copy the application "
30295                         echo "[$0]   $BASENAME inside the floppy "
30296                         echo "[$0]   disk image!"
30297                         break
30298                     fi
30299                 fi
30300             fi
30301         done
30302         sync
30303         #
30304         # Collegamento delle applicazioni più semplici,
30305         # provenienti da altri sistemi operativi.
30306         #
30307         for o in `find ported/mix -name \*.o -print`
30308         do
30309             if    [ "$o" = "ported/mix/crt0.o" ] \
30310                || [ ! -e "$o" ] \
30311                || echo "$o" | grep ".crt0.o$" > /dev/null
30312             then
30313                 #
30314                 # Il file non esiste oppure si tratta di `...crt0.s'.
30315                 #
30316                 true
30317             else
30318                 #
30319                 # File oggetto differente da `...crt0.s'.
30320                 #
30321                 EXEC=`echo "$o" | sed "s/\.o$//"`
30322                 BASENAME=`basename $o .o`
30323                 if [ -e "ported/mix/$BASENAME.crt0.o" ]
30324                 then
30325                     #
30326                     # Qui c'è un file `...crt0.o' specifico.
30327                     #
30328                     ld86 -i -d -s -o $EXEC \
```



```
30329                          applic/$BASENAME.crt0.o $o $OBJLIB
30330                 else
30331                     #
30332                     # Qui si usa il file `crt0.o' generale.
30333                     #
30334                     ld86 -i -d -s -o $EXEC applic/crt0.o $o $OBJLIB
30335                 fi
30336                 #
30337                 if [ -x "$EXEC" ]
30338                 then
30339                     if mount | grep /mnt/os16.a > /dev/null
30340                     then
30341                         mkdir /mnt/os16.b/bin/ 2> /dev/null
30342                         cp -f "$EXEC" /mnt/os16.b/bin
30343                     else
30344                         echo "[$0] Cannot copy the application "
30345                         echo "[$0]   $EXEC inside the floppy "
30346                         echo "[$0]   disk image!"
30347                         break
30348                     fi
30349                 fi
30350             fi
30351         done
30352         sync
30353         #
30354         # Altre applicazioni più importanti.
30355         #
30356         for d in ported/*
30357         do
30358             if [ -d "$d" ]
30359             then
30360                 #
30361                 #
30362                 #
30363                 OBJECTS=""
30364                 BASENAME=`basename $d`
30365                 EXEC="$d/$BASENAME"
30366                 #
30367                 #
30368                 #
30369                 if [ "$BASENAME" = "mix" ]
30370                 then
30371                     #
30372                     # già fatto.
30373                     #
30374                     continue
30375                 fi
30376                 #
30377                 #
30378                 #
30379                 for o in $d/*.o
30380                 do
30381                     if    [ "$o" = "$d/crt0.o" ] \
30382                        || [ ! -e "$o" ]
30383                     then
30384                         true
30385                     else
30386                         OBJECTS="$OBJECTS $o"
30387                     fi
30388                 done
30389                 #
30390                 ld86 -i -d -s -o $EXEC $d/crt0.o $OBJECTS $OBJLIB
30391                 #
30392                 if [ -x "$d/$BASENAME" ]
30393                 then
30394                     if mount | grep /mnt/os16.b > /dev/null
30395                     then
30396                         mkdir /mnt/os16.b/bin/ 2> /dev/null
30397                         cp -f "$EXEC" /mnt/os16.b/bin
30398                     else
30399                         echo "[$0] Cannot copy the application "
30400                         echo "[$0]   $BASENAME inside the floppy "
30401                         echo "[$0]   disk image!"
30402                         break
30403                     fi
30404                 fi
30405             fi
30406         done
30407         sync
30408
30409     fi
30410 }
30411 #
30412 # Start.
30413 #
30414 if [ -d kernel ]   && \
30415    [ -d applic ]   && \
30416    [ -d lib  ]
30417 then
30418     OS16PATH=`pwd`
30419     main
30420 else
30421     echo "[$0] Running from a wrong directory!"
30422 fi
```

os16: «kernel/devices.h»

«

Si veda la sezione u0.1.

```
40001  #ifndef _KERNEL_DEVICES_H
40002  #define _KERNEL_DEVICES_H 1
40003
40004  #include <sys/os16.h>
40005  #include <sys/types.h>
40006  //-------------------------------------------------------------------------
40007  #define DEV_READ                0
40008  #define DEV_WRITE               1
40009  ssize_t dev_io    (pid_t pid, dev_t device, int rw, off_t offset,
40010                     void *buffer, size_t size, int *eof);
40011  //-------------------------------------------------------------------------
40012  // The following functions are used only by 'dev_io()'.
40013  //-------------------------------------------------------------------------
40014  ssize_t dev_mem   (pid_t pid, dev_t device, int rw, off_t offset,
40015                     void *buffer, size_t size, int *eof);
40016  ssize_t dev_tty   (pid_t pid, dev_t device, int rw, off_t offset,
40017                     void *buffer, size_t size, int *eof);
40018  ssize_t dev_dsk   (pid_t pid, dev_t device, int rw, off_t offset,
40019                     void *buffer, size_t size, int *eof);
40020  ssize_t dev_kmem  (pid_t pid, dev_t device, int rw, off_t offset,
40021                     void *buffer, size_t size, int *eof);
40022  //-------------------------------------------------------------------------
40023
40024  #endif
```

kernel/devices/dev_dsk.c

«

Si veda la sezione i159.1.2.

```
50001  #include <sys/os16.h>
50002  #include <kernel/devices.h>
50003  #include <sys/types.h>
50004  #include <errno.h>
50005  #include <kernel/memory.h>
50006  #include <kernel/ibm_i86.h>
50007  #include <kernel/proc.h>
50008  #include <string.h>
50009  #include <signal.h>
50010  #include <kernel/k_libc.h>
50011  #include <ctype.h>
50012  #include <kernel/tty.h>
50013  //-------------------------------------------------------------------------
50014  ssize_t
50015  dev_dsk (pid_t pid, dev_t device, int rw, off_t offset, void *buffer,
50016           size_t size, int *eof)
50017  {
50018      ssize_t n;
50019      int     dev_minor = minor (device);
50020
50021      if (rw == DEV_READ)
50022        {
50023          n = dsk_read_bytes (dev_minor, offset, buffer, size);
50024        }
50025      else
50026        {
50027          n = dsk_write_bytes (dev_minor, offset, buffer, size);
50028        }
50029      return (n);
50030  }
```

kernel/devices/dev_io.c

«

Si veda la sezione i159.1.1.

```
60001  #include <sys/os16.h>
60002  #include <kernel/devices.h>
60003  #include <sys/types.h>
60004  #include <errno.h>
60005  #include <kernel/memory.h>
60006  #include <kernel/ibm_i86.h>
60007  #include <kernel/proc.h>
60008  #include <string.h>
60009  #include <signal.h>
60010  #include <kernel/k_libc.h>
60011  #include <ctype.h>
60012  #include <kernel/tty.h>
60013  //-------------------------------------------------------------------------
60014  ssize_t
60015  dev_io (pid_t pid, dev_t device, int rw, off_t offset,
60016          void *buffer, size_t size, int *eof)
60017  {
60018      int dev_major = major (device);
60019      if (rw != DEV_READ && rw != DEV_WRITE)
60020        {
60021          errset (EIO);
60022          return (-1);
60023        }
60024      switch (dev_major)
60025        {
60026          case DEV_MEM_MAJOR:
60027              return (dev_mem (pid, device, rw, offset, buffer, size,
60028                               eof));
60029          case DEV_TTY_MAJOR:
60030              return (dev_tty (pid, device, rw, offset, buffer, size,
60031                               eof));
60032          case DEV_CONSOLE_MAJOR:
60033              return (dev_tty (pid, device, rw, offset, buffer, size,
60034                               eof));
60035          case DEV_DSK_MAJOR:
60036              return (dev_dsk (pid, device, rw, offset, buffer, size,
60037                               eof));
60038          case DEV_KMEM_MAJOR:
60039              return (dev_kmem (pid, device, rw, offset, buffer, size,
60040                                eof));
60041          default:
60042              errset (ENODEV);
60043              return (-1);
60044        }
60045  }
```


kernel/devices/dev_kmem.c

«

Si veda la sezione i159.1.3.

```
70001  #include <sys/os16.h>
70002  #include <kernel/devices.h>
70003  #include <sys/types.h>
70004  #include <errno.h>
70005  #include <kernel/memory.h>
70006  #include <kernel/ibm_i86.h>
70007  #include <kernel/proc.h>
70008  #include <string.h>
70009  #include <signal.h>
70010  #include <kernel/k_libc.h>
70011  #include <ctype.h>
70012  #include <kernel/tty.h>
70013  //-------------------------------------------------------------------------
70014  ssize_t
70015  dev_kmem (pid_t pid, dev_t device, int rw, off_t offset, void *buffer,
70016            size_t size, int *eof)
70017  {
70018    size_t    size_real;
70019    inode_t  *inode;
70020    sb_t     *sb;
70021    file_t   *file;
70022    void     *start;
70023    //
70024    // Only read is allowed.
70025    //
70026    if (rw != DEV_READ)
70027      {
70028        errset (EIO);                           // I/O error.
70029        return ((ssize_t) -1);
70030      }
70031    //
70032    // Only positive offset is allowed.
70033    //
70034    if (offset < 0)
70035      {
70036        errset (EIO);                           // I/O error.
70037        return ((ssize_t) -1);
70038      }
70039    //
70040    // Read is selected (and is the only access allowed).
70041    //
70042    switch (device)
70043      {
70044        case DEV_KMEM_PS:
70045            //
70046            // Verify if the selected slot can be read.
70047            //
70048            if (offset >= PROCESS_MAX)
70049              {
70050                errset (EIO);                   // I/O error.
70051                return ((ssize_t) -1);
70052              }
70053            //
70054            // Correct the size to be read.
70055            //
70056            if (sizeof (proc_t) < size)
70057              {
70058                size = sizeof (proc_t);
70059              }
70060            //  //
70061            //  // Correct the size to be read.
70062            //  //
70063            //  size_real = ((sizeof (proc_t)) * (PROCESS_MAX - offset));
70064            //  if (size_real < size)
70065            //    {
70066            //      size = size_real;
70067            //    }
70068            //
70069            // Get the pointer to the selected slot.
70070            //
70071            start = proc_reference ((pid_t) offset);
70072            break;
70073        case DEV_KMEM_MMP:
70074            //
70075            // Correct the size to be read.
70076            //
70077            size_real = (MEM_MAX_BLOCKS/8);
70078            if (size_real < size)
70079              {
70080                size = size_real;
70081              }
70082            //
70083            // Get the pointer to the map.
```

```
70084           //
70085           start = mb_reference ();
70086           break;
70087       case DEV_KMEM_SB:
70088           //
70089           // Get a reference to the super block table.
70090           //
70091           sb = sb_reference (0);
70092           //
70093           // Correct the size to be read.
70094           //
70095           if (sizeof (sb_t) < size)
70096             {
70097               size = sizeof (sb_t);
70098             }
70099           //
70100           // Get the pointer to the selected super block slot.
70101           //
70102           start = &sb[offset];
70103           break;
70104       case DEV_KMEM_INODE:
70105           //
70106           // Get a reference to the inode table.
70107           //
70108           inode = inode_reference (0, 0);
70109           //
70110           // Correct the size to be read.
70111           //
70112           if (sizeof (inode_t) < size)
70113             {
70114               size = sizeof (inode_t);
70115             }
70116           //
70117           // Get the pointer to the selected inode slot.
70118           //
70119           start = &inode[offset];
70120           break;
70121       case DEV_KMEM_FILE:
70122           //
70123           // Get a reference to the file table.
70124           //
70125           file = file_reference (0);
70126           //
70127           // Correct the size to be read.
70128           //
70129           if (sizeof (file_t) < size)
70130             {
70131               size = sizeof (file_t);
70132             }
70133           //
70134           // Get the pointer to the selected inode slot.
70135           //
70136           start = &file[offset];
70137           break;
70138       default:
70139           errset (ENODEV);            // No such device.
70140           return ((ssize_t) -1);
70141     }
70142   //
70143   // At this point, data is ready to be copied to the buffer.
70144   //
70145   memcpy (buffer, start, size);
70146   //
70147   // Return size read.
70148   //
70149   return (size);
70150 }
```

kernel/devices/dev_mem.c

«

Si veda la sezione i159.1.4.

```
80001 #include <sys/os16.h>
80002 #include <kernel/devices.h>
80003 #include <sys/types.h>
80004 #include <errno.h>
80005 #include <kernel/memory.h>
80006 #include <kernel/ibm_i86.h>
80007 #include <kernel/proc.h>
80008 #include <string.h>
80009 #include <signal.h>
80010 #include <kernel/k_libc.h>
80011 #include <ctype.h>
80012 #include <kernel/tty.h>
80013 //------------------------------------------------------------------------
80014 ssize_t
80015 dev_mem (pid_t pid, dev_t device, int rw, off_t offset, void *buffer,
80016          size_t size, int *eof)
80017 {
80018     uint8_t  *buffer08 = (uint8_t *) buffer;
80019     uint16_t *buffer16 = (uint16_t *) buffer;
80020     ssize_t n;
80021
80022     if      (device == DEV_MEM)                         // DEV_MEM
80023       {
80024         if (rw == DEV_READ)
80025           {
80026             n = mem_read ((addr_t) offset, buffer, size);
80027           }
80028         else
80029           {
```



```
80030             n = mem_write ((addr_t) offset, buffer, size);
80031           }
80032       }
80033     else if (device == DEV_NULL)                        // DEV_NULL
80034       {
80035         n = 0;
80036       }
80037     else if (device == DEV_ZERO)                        // DEV_ZERO
80038       {
80039         if (rw == DEV_READ)
80040           {
80041             for (n = 0; n < size; n++)
80042               {
80043                 buffer08[n] = 0;
80044               }
80045           }
80046         else
80047           {
80048             n = 0;
80049           }
80050       }
80051     else if (device == DEV_PORT)                        // DEV_PORT
80052       {
80053         if (rw == DEV_READ)
80054           {
80055             if (size == 1)
80056               {
80057                 buffer08[0] = in_8 (offset);
80058                 n = 1;
80059               }
80060             else if (size == 2)
80061               {
80062                 buffer16[0] = in_16 (offset);
80063                 n = 2;
80064               }
80065             else
80066               {
80067                 n = 0;
80068               }
80069           }
80070         else
80071           {
80072             if (size == 1)
80073               {
80074                 out_8 (offset, buffer08[0]);
80075               }
80076             else if (size == 2)
80077               {
80078                 out_16 (offset, buffer16[0]);
80079                 n = 2;
80080               }
80081             else
80082               {
80083                 n = 0;
80084               }
80085           }
80086       }
80087     else
80088       {
80089         errset (ENODEV);
80090         return (-1);
80091       }
80092     return (n);
80093 }
```

kernel/devices/dev_tty.c

«

Si veda la sezione i159.1.5.

```
90001 #include <sys/os16.h>
90002 #include <kernel/devices.h>
90003 #include <sys/types.h>
90004 #include <errno.h>
90005 #include <kernel/memory.h>
90006 #include <kernel/ibm_i86.h>
90007 #include <kernel/proc.h>
90008 #include <string.h>
90009 #include <signal.h>
90010 #include <kernel/k_libc.h>
90011 #include <ctype.h>
90012 #include <kernel/tty.h>
90013 //------------------------------------------------------------------------
90014 ssize_t
90015 dev_tty (pid_t pid, dev_t device, int rw, off_t offset, void *buffer,
90016          size_t size, int *eof)
90017 {
90018     uint8_t *buffer08 = (uint8_t *) buffer;
90019     ssize_t  n;
90020     proc_t  *ps;
90021     //
90022     // Get process. Variable 'ps' will be 'NULL' if the process ID is
90023     // not valid.
90024     //
90025     ps = proc_reference (pid);
90026     //
90027     // Convert 'DEV_TTY' with the controlling terminal for the process.
90028     //
90029     if (device == DEV_TTY)
90030       {
90031         device = ps->device_tty;
90032         //
```

```
90033         // As a last resort, use the generic 'DEV_CONSOLE'.
90034         //
90035         if (device == 0 || device == DEV_TTY)
90036           {
90037             device = DEV_CONSOLE;
90038           }
90039       }
90040     //
90041     // Convert 'DEV_CONSOLE' to the currently active console.
90042     //
90043     if (device == DEV_CONSOLE)
90044       {
90045         device = tty_console ((dev_t) 0);
90046         //
90047         // As a last resort, use the first console: 'DEV_CONSOLE0'.
90048         //
90049         if (device == 0 || device == DEV_TTY)
90050           {
90051             device = DEV_CONSOLE0;
90052           }
90053       }
90054     //
90055     // Read or write.
90056     //
90057     if (rw == DEV_READ)
90058       {
90059         for (n = 0; n < size; n++)
90060           {
90061             buffer08[n] = tty_read (device);
90062             if (buffer08[n] == 0)
90063               {
90064                 //
90065                 // If the pid is not the kernel, should put the process
90066                 // to sleep, waiting for the key.
90067                 //
90068                 if (pid == 0 || ps == NULL)
90069                   {
90070                     //
90071                     // For the kernel there is no sleep and for an
90072                     // unidentified process, either.
90073                     //
90074                     break;
90075                   }
90076                 //
90077                 // Put the process to sleep.
90078                 //
90079                 ps->status         = PROC_SLEEPING;
90080                 ps->ret            = 0;
90081                 ps->wakeup_events  = WAKEUP_EVENT_TTY;
90082                 ps->wakeup_signal  = 0;
90083                 ps->wakeup_timer   = 0;
90084                 //
90085                 break;
90086               }
90087             //
90088             // Check for control characters.
90089             //
90090             if (buffer08[n] == 0x04)    // EOT
90091               {
90092                 //
90093                 // Return EOF.
90094                 //
90095                 *eof = 1;
90096                 break;
90097               }
90098             //
90099             // At this point, show the character on screen, even if it
90100             // is not nice. It is necessary to show something, because
90101             // the tty handling is very poor and the library for line
90102             // input, calculate cursor position based on the characters
90103             // received.
90104             //
90105             tty_write (device, (int) buffer08[n]);
90106           }
90107       }
90108     else
90109       {
90110         for (n = 0; n < size; n++)
90111           {
90112             tty_write (device, (int) buffer08[n]);
90113           }
90114       }
90115     return (n);
90116 }
```

os16: «kernel/diag.h»

«

Si veda la sezione u0.2.

```
100001 #ifndef _KERNEL_DIAG_H
100002 #define _KERNEL_DIAG_H 1
100003
100004 #include <stdint.h>
100005 #include <kernel/fs.h>
100006 #include <sys/types.h>
100007 #include <kernel/proc.h>
100008
100009 //-------------------------------------------------------------------------
100010 uint8_t  reverse_8_bit  (uint8_t  source);
100011 uint16_t reverse_16_bit (uint16_t source);
100012 uint32_t reverse_32_bit (uint32_t source);
```



```
100013
100014 #define reverse_char(s)      ((char)     reverse_8_bit  ((uint8_t)  s))
100015 #define reverse_short(s)     ((short)    reverse_16_bit ((uint16_t) s))
100016 #define reverse_int(s)       ((int)      reverse_16_bit ((uint16_t) s))
100017 #define reverse_long(s)      ((long)     reverse_32_bit ((uint32_t) s))
100018 #define reverse_long_int(s)  ((long int) reverse_32_bit ((uint32_t) s))
100019 //-------------------------------------------------------------------------
100020 void print_hex_8  (void *data, size_t elements);
100021 void print_hex_16 (void *data, size_t elements);
100022 void print_hex_32 (void *data, size_t elements);
100023
100024 #define print_hex_char(d, e)      (print_hex_8  (d, e))
100025 #define print_hex_short(d, e)     (print_hex_16 (d, e))
100026 #define print_hex_int(d, e)       (print_hex_16 (d, e))
100027 #define print_hex_long(d, e)      (print_hex_32 (d, e))
100028 #define print_hex_long_int(d, e)  (print_hex_32 (d, e))
100029 //-------------------------------------------------------------------------
100030 void print_hex_8_reverse  (void *data, size_t elements);
100031 void print_hex_16_reverse (void *data, size_t elements);
100032 void print_hex_32_reverse (void *data, size_t elements);
100033
100034 #define print_hex_char_reverse(d, e)      (print_hex_8_reverse  (d, e))
100035 #define print_hex_short_reverse(d, e)     (print_hex_16_reverse (d, e))
100036 #define print_hex_int_reverse(d, e)       (print_hex_16_reverse (d, e))
100037 #define print_hex_long_reverse(d, e)      (print_hex_32_reverse (d, e))
100038 #define print_hex_long_int_reverse(d, e)  (print_hex_32_reverse (d, e))
100039 //-------------------------------------------------------------------------
100040 void print_segments (void);
100041 void print_kmem     (void);
100042 //-------------------------------------------------------------------------
100043 void print_mb_map     (void);
100044 void print_memory_map (void);
100045 //-------------------------------------------------------------------------
100046 void print_superblock       (sb_t *sb);
100047 void print_inode            (inode_t *inode);
100048 void print_inode_map        (sb_t *sb, uint16_t *bitmap);
100049 void print_zone_map         (sb_t *sb, uint16_t *bitmap);
100050 void print_inode_head       (void);
100051 void print_inode_list       (void);
100052 void print_inode_zones_head (void);
100053 void print_inode_zones      (inode_t *inode);
100054 void print_inode_zones_list (void);
100055 //-------------------------------------------------------------------------
100056 void print_proc_head (void);
100057 void print_proc_pid  (proc_t *ps, pid_t pid);
100058 void print_proc_list (void);
100059 //-------------------------------------------------------------------------
100060 void print_file_head (void);
100061 void print_file_num  (int num);
100062 void print_file_list (void);
100063 //-------------------------------------------------------------------------
100064 void print_fd_head (void);
100065 void print_fd      (fd_t *fd);
100066 void print_fd_list (pid_t pid);
100067 //-------------------------------------------------------------------------
100068 void print_time      (void);
100069 //-------------------------------------------------------------------------
100070
100071 #endif
```

kernel/diag/print_fd.c

«

Si veda la sezione u0.2.

```
110001 #include <sys/os16.h>
110002 #include <kernel/diag.h>
110003 #include <kernel/k_libc.h>
110004 #include <fcntl.h>
110005 //-------------------------------------------------------------------------
110006 void
110007 print_fd (fd_t *fd)
110008 {
110009     k_printf ("%04x %6li %3i %c/%c %05o %5i %3i %5li %4i %04x %3i",
110010         (unsigned int)      fd->fl_flags,
110011         (unsigned long int) fd->file->offset,
110012         (unsigned int)      fd->file->references,
110013         (fd->file->oflags & O_RDONLY ? 'r' : ' '),
110014         (fd->file->oflags & O_WRONLY ? 'w' : ' '),
110015         (unsigned int)      fd->file->inode->mode,
110016         (unsigned int)      fd->file->inode->uid,
110017         (unsigned int)      fd->file->inode->gid,
110018         (unsigned long int) fd->file->inode->size,
110019         (unsigned int)      fd->file->inode->links,
110020         (unsigned int)      fd->file->inode->sb->device,
110021         (unsigned int)      fd->file->inode->ino);
110022     k_printf ("\n");
110023 }
```

kernel/diag/print_fd_head.c

«

Si veda la sezione u0.2.

```
120001 #include <sys/os16.h>
120002 #include <kernel/diag.h>
120003 #include <kernel/k_libc.h>
120004 //-------------------------------------------------------------------------
120005 void
120006 print_fd_head (void)
120007 {
120008
```

```
120009      k_printf ("n. stat offset ref flg  mode   uid gid  size lnks  ");
120010      k_printf ("dev ino\n");
120011  }
```

kernel/diag/print_fd_list.c

«

Si veda la sezione u0.2.

```
130001  #include <sys/os16.h>
130002  #include <kernel/diag.h>
130003  #include <kernel/k_libc.h>
130004  //-----------------------------------------------------------------------
130005  void
130006  print_fd_list (pid_t pid)
130007  {
130008      int      fdn = 0;
130009      fd_t *fd;
130010      fd = fd_reference (pid, &fdn);
130011      print_fd_head ();
130012      for (fdn = 0; fdn < OPEN_MAX; fdn++)
130013        {
130014          if (fd[fdn].file != NULL)
130015            {
130016              k_printf ("%2i ", fdn);
130017              print_fd (fd);
130018            }
130019        }
130020  }
```

kernel/diag/print_file_head.c

«

Si veda la sezione u0.2.

```
140001  #include <sys/os16.h>
140002  #include <kernel/diag.h>
140003  #include <kernel/k_libc.h>
140004  //-----------------------------------------------------------------------
140005  void
140006  print_file_head (void)
140007  {
140008
140009      k_printf ("n. ref flg  mode uid  size lnks  dev ino\n");
140010  }
```

kernel/diag/print_file_list.c

«

Si veda la sezione u0.2.

```
150001  #include <sys/os16.h>
150002  #include <kernel/diag.h>
150003  #include <kernel/k_libc.h>
150004  //-----------------------------------------------------------------------
150005  void
150006  print_file_list (void)
150007  {
150008      int     fno;
150009      file_t *file = file_reference (0);
150010      //
150011      print_file_head ();
150012      //
150013      for (fno = 0; fno < FILE_MAX_SLOTS; fno++)
150014        {
150015          if (file[fno].references > 0)
150016            {
150017              print_file_num (fno);
150018            }
150019        }
150020  }
```

kernel/diag/print_file_num.c

«

Si veda la sezione u0.2.

```
160001  #include <sys/os16.h>
160002  #include <kernel/diag.h>
160003  #include <kernel/k_libc.h>
160004  #include <fcntl.h>
160005  //-----------------------------------------------------------------------
160006  void
160007  print_file_num (int fno)
160008  {
160009      file_t *file = file_reference (fno);
160010
160011      k_printf ("%2i %3i %c/%c %05o %3i %5li %4i %04x %3i",
160012                (unsigned int)      fno,
160013                (unsigned int)      file->references,
160014                (file->oflags & O_RDONLY ? 'r' : ' '),
160015                (file->oflags & O_WRONLY ? 'w' : ' '),
160016                (unsigned int)      file->inode->mode,
160017                (unsigned int)      file->inode->uid,
160018                (unsigned long int) file->inode->size,
160019                (unsigned int)      file->inode->links,
160020                (unsigned int)      file->inode->sb->device,
160021                (unsigned int)      file->inode->ino);
160022      k_printf ("\n");
160023  }
```


kernel/diag/print_hex_16.c

«

Si veda la sezione u0.2.

```
170001  #include <sys/os16.h>
170002  #include <kernel/diag.h>
170003  #include <kernel/k_libc.h>
170004  #include <inttypes.h>
170005  #include <stdio.h>
170006  //-----------------------------------------------------------------------
170007  void
170008  print_hex_16 (void *data, size_t elements)
170009  {
170010      uint16_t *element = (uint16_t *) data;
170011      int i;
170012      for (i = 0; i < elements; i++)
170013        {
170014          k_printf ("%04" PRIx16, (uint16_t) element[i]);
170015        }
170016  }
```

kernel/diag/print_hex_16_reverse.c

«

Si veda la sezione u0.2.

```
180001  #include <sys/os16.h>
180002  #include <kernel/diag.h>
180003  #include <kernel/k_libc.h>
180004  #include <inttypes.h>
180005  #include <stdio.h>
180006  //-----------------------------------------------------------------------
180007  void
180008  print_hex_16_reverse (void *data, size_t elements)
180009  {
180010      uint16_t *element = (uint16_t *) data;
180011      int i;
180012      for (i = 0; i < elements; i++)
180013        {
180014          k_printf ("%04" PRIx16, reverse_16_bit (element[i]));
180015        }
180016  }
```

kernel/diag/print_hex_32.c

«

Si veda la sezione u0.2.

```
190001  #include <sys/os16.h>
190002  #include <kernel/diag.h>
190003  #include <kernel/k_libc.h>
190004  #include <inttypes.h>
190005  #include <stdio.h>
190006  //-----------------------------------------------------------------------
190007  void
190008  print_hex_32 (void *data, size_t elements)
190009  {
190010      uint32_t *element = (uint32_t *) data;
190011      int i;
190012      for (i = 0; i < elements; i++)
190013        {
190014          k_printf ("%08" PRIx32, (uint32_t) element[i]);
190015        }
190016  }
```

kernel/diag/print_hex_32_reverse.c

«

Si veda la sezione u0.2.

```
200001  #include <sys/os16.h>
200002  #include <kernel/diag.h>
200003  #include <kernel/k_libc.h>
200004  #include <inttypes.h>
200005  #include <stdio.h>
200006  //-----------------------------------------------------------------------
200007  void
200008  print_hex_32_reverse (void *data, size_t elements)
200009  {
200010      uint32_t *element = (uint32_t *) data;
200011      int i;
200012      for (i = 0; i < elements; i++)
200013        {
200014          k_printf ("%08" PRIx32, reverse_32_bit (element[i]));
200015        }
200016  }
```

kernel/diag/print_hex_8.c

«

Si veda la sezione u0.2.

```
210001  #include <sys/os16.h>
210002  #include <kernel/diag.h>
210003  #include <kernel/k_libc.h>
210004  #include <inttypes.h>
210005  #include <stdio.h>
210006  //-----------------------------------------------------------------------
210007  void
210008  print_hex_8 (void *data, size_t elements)
210009  {
210010      uint8_t *element = (uint8_t *) data;
```

```
210011      int i;
210012      for (i = 0; i < elements; i++)
210013        {
210014          k_printf ("%02" PRIx8, (uint16_t) element[i]);
210015        }
210016  }
```

kernel/diag/print_hex_8_reverse.c

«

Si veda la sezione u0.2.

```
220001  #include <sys/os16.h>
220002  #include <kernel/diag.h>
220003  #include <kernel/k_libc.h>
220004  #include <inttypes.h>
220005  #include <stdio.h>
220006  //---------------------------------------------------------------------
220007  void
220008  print_hex_8_reverse (void *data, size_t elements)
220009  {
220010      uint8_t *element = (uint8_t *) data;
220011      int i;
220012      for (i = 0; i < elements; i++)
220013        {
220014          k_printf ("%02" PRIx8, reverse_8_bit (element[i]));
220015        }
220016  }
```

kernel/diag/print_inode.c

«

Si veda la sezione u0.2.

```
230001  #include <sys/os16.h>
230002  #include <kernel/diag.h>
230003  #include <kernel/k_libc.h>
230004  //---------------------------------------------------------------------
230005  void
230006  print_inode (inode_t *inode)
230007  {
230008      unsigned long int seconds;
230009      unsigned long int seconds_d;
230010      unsigned long int seconds_h;
230011      unsigned int      d;
230012      unsigned int      h;
230013      unsigned int      m;
230014      unsigned int      s;
230015      dev_t             device_attached = 0;
230016      //
230017      if (inode == NULL)
230018        {
230019          return;
230020        }
230021      //
230022      seconds    = inode->time;
230023      d          = seconds / 86400L;              // 24 * 60 * 60
230024      seconds_d  = d;
230025      seconds_d *= 86400;
230026      seconds   -= seconds_d;
230027      h          = seconds / 3840;                // 60 * 60
230028      seconds_h  = h;
230029      seconds_h *= 3840;
230030      seconds   -= seconds_h;
230031      m          = seconds / 60;
230032      s          = seconds % 60;
230033      //
230034      if (inode->sb_attached != NULL)
230035        {
230036          device_attached = inode->sb_attached->device;
230037        }
230038      //
230039      k_printf ("%04x %4i %3i %c %4x %06o %4i %3i %7li ",
230040                (unsigned int) inode->sb->device,
230041                (unsigned int) inode->ino,
230042                (unsigned int) inode->references,
230043                (inode->changed ? '!' : ' '),
230044                (unsigned int) device_attached,
230045                (unsigned int) inode->mode,
230046                (unsigned int) inode->uid,
230047                (unsigned int) inode->gid,
230048                (unsigned long int) inode->size);
230049
230050      k_printf ("%5i %2i:%02i:%02i %3i\n",
230051                d, h, m, s,
230052                (unsigned int) inode->links);
230053
230054  }
```

kernel/diag/print_inode_head.c

«

Si veda la sezione u0.2.

```
240001  #include <sys/os16.h>
240002  #include <kernel/diag.h>
240003  #include <kernel/k_libc.h>
240004  //---------------------------------------------------------------------
240005  void
240006  print_inode_head (void)
240007  {
```



```
240008      k_printf (" dev  ino ref c mntd   mode  uid gid    ");
240009      k_printf ("size  date     time lnk \n");
240010  }
```

kernel/diag/print_inode_list.c

«

Si veda la sezione u0.2.

```
250001  #include <sys/os16.h>
250002  #include <kernel/diag.h>
250003  #include <kernel/k_libc.h>
250004  //---------------------------------------------------------------------
250005  void
250006  print_inode_list (void)
250007  {
250008      ino_t ino;
250009      inode_t *inode = inode_reference (0, 0);
250010      print_inode_head ();
250011      for (ino = 0; ino < INODE_MAX_SLOTS; ino++)
250012        {
250013          if (inode[ino].references > 0)
250014            {
250015              print_inode (&inode[ino]);
250016            }
250017        }
250018  }
```

kernel/diag/print_inode_map.c

«

Si veda la sezione u0.2.

```
260001  #include <sys/os16.h>
260002  #include <kernel/diag.h>
260003  #include <kernel/k_libc.h>
260004  //---------------------------------------------------------------------
260005  void
260006  print_inode_map (sb_t *sb, uint16_t *bitmap)
260007  {
260008      size_t size;
260009      if (sb->inodes % 16)
260010        {
260011          size = sb->inodes/16 + 1;
260012        }
260013      else
260014        {
260015          size = sb->inodes/16;
260016        }
260017      print_hex_16_reverse (bitmap, size);
260018  }
```

kernel/diag/print_inode_zone_list.c

«

Si veda la sezione u0.2.

```
270001  #include <sys/os16.h>
270002  #include <kernel/diag.h>
270003  #include <kernel/k_libc.h>
270004  //---------------------------------------------------------------------
270005  void
270006  print_inode_zones_list (void)
270007  {
270008      ino_t ino;
270009      inode_t *inode = inode_reference (0, 0);
270010      print_inode_zones_head ();
270011      for (ino = 0; ino < INODE_MAX_SLOTS; ino++)
270012        {
270013          if (inode[ino].references > 0)
270014            {
270015              print_inode_zones (&inode[ino]);
270016            }
270017        }
270018  }
```

kernel/diag/print_inode_zones.c

«

Si veda la sezione u0.2.

```
280001  #include <sys/os16.h>
280002  #include <kernel/diag.h>
280003  #include <kernel/k_libc.h>
280004  //---------------------------------------------------------------------
280005  void
280006  print_inode_zones (inode_t *inode)
280007  {
280008      int i;
280009      //
280010      if (inode == NULL)
280011        {
280012          return;
280013        }
280014      //
280015      k_printf ("%04x %4i ",
280016                (unsigned int) inode->sb->device,
280017                (unsigned int) inode->ino);
280018
280019      for (i = 0; i < 7; i++)
280020        {
```

```
280021          if (inode->direct[i] != 0)
280022            {
280023              k_printf ("%04x ", (unsigned int) inode->direct[i]);
280024            }
280025          else
280026            {
280027              k_printf ("     ");
280028            }
280029        }
280030      if (inode->indirect1 != 0)
280031        {
280032          k_printf ("%04x ",     (unsigned int) inode->indirect1);
280033        }
280034      else
280035        {
280036          k_printf ("     ");
280037        }
280038      if (inode->indirect2 != 0)
280039        {
280040          k_printf ("%04x",      (unsigned int) inode->indirect2);
280041        }
280042      else
280043        {
280044          k_printf ("    ");
280045        }
280046      k_printf ("\n");
280047  }
```

kernel/diag/print_inode_zones_head.c

«

Si veda la sezione u0.2.

```
290001  #include <sys/os16.h>
290002  #include <kernel/diag.h>
290003  #include <kernel/k_libc.h>
290004  //---------------------------------------------------------------------
290005  void
290006  print_inode_zones_head (void)
290007  {
290008      k_printf (" dev  ino zn_0 zn_1 zn_2 zn_3 zn_4 zn_5 zn_6 ");
290009      k_printf ("ind1 ind2\n");
290010  }
```

kernel/diag/print_kmem.c

«

Si veda la sezione u0.2.

```
300001  #include <sys/os16.h>
300002  #include <kernel/diag.h>
300003  #include <kernel/k_libc.h>
300004  //---------------------------------------------------------------------
300005  extern uint16_t _ksp;
300006  extern uint16_t _etext;
300007  extern uint16_t _edata;
300008  extern uint16_t _end;
300009  //---------------------------------------------------------------------
300010  void
300011  print_kmem (void)
300012  {
300013      k_printf ("etext=%04x edata=%04x ebss=%04x ksp=%04x",
300014                (unsigned int) &_etext,
300015                (unsigned int) &_edata, (unsigned int) &_end, _ksp);
300016  }
```

kernel/diag/print_mb_map.c

«

Si veda la sezione u0.2.

```
310001  #include <sys/os16.h>
310002  #include <kernel/diag.h>
310003  #include <kernel/k_libc.h>
310004  //---------------------------------------------------------------------
310005  void
310006  print_mb_map (void)
310007  {
310008      uint16_t    *mb = mb_reference ();
310009      unsigned int i;
310010      for (i = 0; i < (MEM_MAX_BLOCKS / 16); i++)
310011        {
310012          k_printf ("%04x", mb[i]);
310013        }
310014  }
```

kernel/diag/print_memory_map.c

«

Si veda la sezione u0.2.

```
320001  #include <sys/os16.h>
320002  #include <kernel/diag.h>
320003  #include <kernel/k_libc.h>
320004  //---------------------------------------------------------------------
320005  void
320006  print_memory_map (void)
320007  {
320008      uint16_t    *mem_block[MEM_BLOCK_SIZE/2];
320009      uint16_t     block_rank;
320010      unsigned int b;
```



```
320011      unsigned int m;
320012      unsigned int i;
320013      addr_t       start;
320014
320015      start = 0;
320016      dev_io ((pid_t) -1, DEV_MEM, DEV_READ, start, mem_block,
320017              MEM_BLOCK_SIZE, NULL);
320018
320019      for (m = 0; m < MEM_MAX_BLOCKS; m++)
320020        {
320021          i = m % 16;
320022          if (i == 0)
320023            {
320024              block_rank = 0;
320025            }
320026          //
320027          for (b = 0; b < (MEM_BLOCK_SIZE / 2); b++)
320028            {
320029              if (mem_block[b])
320030                {
320031                  block_rank |= (0x8000 >> i);
320032                  break;
320033                }
320034            }
320035          //
320036          if (i == 15)
320037            {
320038              k_printf ("%04x", block_rank);
320039            }
320040          //
320041          start += MEM_BLOCK_SIZE;
320042          dev_io ((pid_t) -1, DEV_MEM, DEV_READ, start, mem_block,
320043                  MEM_BLOCK_SIZE, NULL);
320044        }
320045  }
```

kernel/diag/print_proc_head.c

«

Si veda la sezione u0.2.

```
330001  #include <sys/os16.h>
330002  #include <kernel/diag.h>
330003  #include <kernel/k_libc.h>
330004  //---------------------------------------------------------------------
330005  void
330006  print_proc_head (void)
330007  {
330008      k_printf (
330009  "pp  p pg                                                    \n"
330010  "id id rp  tty  uid euid suid usage s iaddr isiz daddr dsiz sp   name\n"
330011            );
330012  }
```

kernel/diag/print_proc_list.c

«

Si veda la sezione u0.2.

```
340001  #include <sys/os16.h>
340002  #include <kernel/diag.h>
340003  #include <kernel/k_libc.h>
340004  //---------------------------------------------------------------------
340005  void
340006  print_proc_list (void)
340007  {
340008      pid_t   pid;
340009      proc_t *ps;
340010      //
340011      print_proc_head ();
340012      //
340013      for (pid = 0; pid < PROCESS_MAX; pid++)
340014        {
340015          ps = proc_reference (pid);
340016          if (ps != NULL && ps->status > 0)
340017            {
340018              print_proc_pid (ps, pid);
340019            }
340020        }
340021  }
```

kernel/diag/print_proc_pid.c

«

Si veda la sezione u0.2.

```
350001  #include <sys/os16.h>
350002  #include <kernel/diag.h>
350003  #include <kernel/k_libc.h>
350004  //---------------------------------------------------------------------
350005  void
350006  print_proc_pid (proc_t *ps, pid_t pid)
350007  {
350008      char  stat;
350009      switch (ps->status)
350010        {
350011          case PROC_EMPTY   : stat = '-'; break;
350012          case PROC_CREATED : stat = 'c'; break;
350013          case PROC_READY   : stat = 'r'; break;
350014          case PROC_RUNNING : stat = 'R'; break;
350015          case PROC_SLEEPING: stat = 's'; break;
```

```
350016          case PROC_ZOMBIE  : stat = 'z'; break;
350017          default           : stat = '?'; break;
350018        }
350019
350020      k_printf ("%2i %2i %2i %04x %4i %4i %4i %02i.%02i %c %05lx %04x ",
350021                (unsigned int) ps->ppid,
350022                (unsigned int) pid,
350023                (unsigned int) ps->pgrp,
350024                (unsigned int) ps->device_tty,
350025                (unsigned int) ps->uid,
350026                (unsigned int) ps->euid,
350027                (unsigned int) ps->suid,
350028                (unsigned int) ((ps->usage / CLOCKS_PER_SEC) / 60),
350029                (unsigned int) ((ps->usage / CLOCKS_PER_SEC) % 60),
350030                stat,
350031                (unsigned long int) ps->address_i,
350032                (unsigned int) ps->size_i);
350033
350034      k_printf ("%05lx %04x %04x %s",
350035                (unsigned long int) ps->address_d,
350036                (unsigned int) ps->size_d,
350037                (unsigned int) ps->sp,
350038                               ps->name);
350039
350040      k_printf ("\n");
350041  }
```

kernel/diag/print_segments.c

«

Si veda la sezione u0.2.

```
360001  #include <sys/os16.h>
360002  #include <kernel/diag.h>
360003  #include <kernel/k_libc.h>
360004  //-----------------------------------------------------------------------
360005  void
360006  print_segments (void)
360007  {
360008      k_printf ("CS=%04x DS=%04x SS=%04x ES=%04x BP=%04x SP=%04x ",
360009                cs (), ds (), ss (), es (), bp (), sp ());
360010      k_printf ("heap_min=%04x", heap_min ());
360011  }
```

kernel/diag/print_superblock.c

«

Si veda la sezione u0.2.

```
370001  #include <sys/os16.h>
370002  #include <kernel/diag.h>
370003  #include <kernel/k_libc.h>
370004  //-----------------------------------------------------------------------
370005  void
370006  print_superblock (sb_t *sb)
370007  {
370008      k_printf ("Inodes:           %i\n", sb->inodes);
370009      k_printf ("Blocks:           %i\n", sb->zones);
370010      k_printf ("First data zone:  %i\n", sb->first_data_zone);
370011      k_printf ("Zone size:        %i\n", (1024 << sb->log2_size_zone));
370012      k_printf ("Max file size:    %li\n", sb->max_file_size);
370013      k_printf ("Inode map blocks: %i\n", sb->map_inode_blocks);
370014      k_printf ("Zone map blocks:  %i\n", sb->map_zone_blocks);
370015  }
```

kernel/diag/print_time.c

«

Si veda la sezione u0.2.

```
380001  #include <sys/os16.h>
380002  #include <kernel/diag.h>
380003  #include <kernel/k_libc.h>
380004  //-----------------------------------------------------------------------
380005  void
380006  print_time (void)
380007  {
380008      unsigned long int ticks   = k_clock ();
380009      unsigned long int seconds = k_time (NULL);
380010      unsigned int      h       = seconds / 60 / 60;
380011      unsigned int      m       = seconds / 60 - h * 60;
380012      unsigned int      s       = seconds - m * 60 - h * 60 * 60;
380013      k_printf ("clock=%08lx, time elapsed=%02u:%02u:%02u",
380014                ticks, h, m, s);
380015  }
```

kernel/diag/print_zone_map.c

«

Si veda la sezione u0.2.

```
390001  #include <sys/os16.h>
390002  #include <kernel/diag.h>
390003  #include <kernel/k_libc.h>
390004  //-----------------------------------------------------------------------
390005  void
390006  print_zone_map (sb_t *sb, uint16_t *bitmap)
390007  {
390008      size_t       size;
390009      unsigned int data_zones = sb->zones - sb->first_data_zone;
390010      if (data_zones % 16)
```



```
390011        {
390012          size = data_zones/16 + 1;
390013        }
390014      else
390015        {
390016          size = data_zones/16;
390017        }
390018      print_hex_16_reverse (bitmap, size);
390019  }
```

kernel/diag/reverse_16_bit.c

«

Si veda la sezione u0.2.

```
400001  #include <sys/os16.h>
400002  #include <kernel/diag.h>
400003  #include <kernel/k_libc.h>
400004  #include <inttypes.h>
400005  //-----------------------------------------------------------------------
400006  uint16_t
400007  reverse_16_bit (uint16_t source)
400008  {
400009      uint16_t destination = 0;
400010      uint16_t mask_src;
400011      uint16_t mask_dst;
400012      int      i;
400013      for (i = 0; i < 16; i++)
400014        {
400015          mask_src = 0x0001 << i;
400016          mask_dst = 0x8000 >> i;
400017          if (source & mask_src)
400018            {
400019              destination |= mask_dst;
400020            }
400021        }
400022      return (destination);
400023  }
```

kernel/diag/reverse_32_bit.c

«

Si veda la sezione u0.2.

```
410001  #include <sys/os16.h>
410002  #include <kernel/diag.h>
410003  #include <kernel/k_libc.h>
410004  //-----------------------------------------------------------------------
410005  uint32_t
410006  reverse_32_bit (uint32_t source)
410007  {
410008      uint32_t destination = 0;
410009      uint32_t mask_src;
410010      uint32_t mask_dst;
410011      int      i;
410012      for (i = 0; i < 32; i++)
410013        {
410014          mask_src = 0x00000001 << i;
410015          mask_dst = 0x80000000 >> i;
410016          if (source & mask_src)
410017            {
410018              destination |= mask_dst;
410019            }
410020        }
410021      return (destination);
410022  }
```

kernel/diag/reverse_8_bit.c

«

Si veda la sezione u0.2.

```
420001  #include <sys/os16.h>
420002  #include <kernel/diag.h>
420003  #include <kernel/k_libc.h>
420004  #include <inttypes.h>
420005  //-----------------------------------------------------------------------
420006  uint8_t
420007  reverse_8_bit (uint8_t source)
420008  {
420009      uint8_t destination = 0;
420010      uint8_t mask_src;
420011      uint8_t mask_dst;
420012      int      i;
420013      for (i = 0; i < 8; i++)
420014        {
420015          mask_src = 0x01 << i;
420016          mask_dst = 0x80 >> i;
420017          if (source & mask_src)
420018            {
420019              destination |= mask_dst;
420020            }
420021        }
420022      return (destination);
420023  }
```

«

Si veda la sezione u0.3.

```
430001  #ifndef _KERNEL_FS_H
430002  #define _KERNEL_FS_H 1
430003
430004  #include <stdint.h>
430005  #include <sys/types.h>
430006  #include <kernel/memory.h>
430007  #include <sys/os16.h>
430008  #include <sys/stat.h>
430009  #include <stdint.h>
430010  #include <const.h>
430011  #include <stdio.h>
430012  #include <limits.h>
430013
430014  //------------------------------------------------------------------------
430015  #define SB_MAX_INODE_BLOCKS          1  // 8192 inodes max.
430016  #define SB_MAX_ZONE_BLOCKS           1  // 8192 data-zones max.
430017  #define SB_BLOCK_SIZE             1024  // Fixed for Minix file system.
430018  #define SB_MAX_ZONE_SIZE          2048  // log2 max is 1.
430019  #define SB_MAP_INODE_SIZE         (SB_MAX_INODE_BLOCKS*512) // [1]
430020  #define SB_MAP_ZONE_SIZE          (SB_MAX_ZONE_BLOCKS*512)  // [1]
430021  //
430022  // [1] blocks * (1024 * 8 / 16) = number of bits, divided 16.
430023  //
430024  //------------------------------------------------------------------------
430025  #define INODE_MAX_INDIRECT_ZONES  (SB_MAX_ZONE_SIZE/2)      // [2]
430026
430027  #define INODE_MAX_REFERENCES      0xFF
430028  //
430029  // [2] number of zone pointers contained inside a zone, used
430030  //     as an indirect inode list (a pointer = 16 bits = 2 bytes).
430031  //
430032  //------------------------------------------------------------------------
430033  typedef uint16_t        zno_t;          // Zone number.
430034  //------------------------------------------------------------------------
430035  // The structured type 'inode_t' must be pre-declared here, because
430036  // the type sb_t, described before the inode structure, has a member
430037  // pointing to a type 'inode_t'. So, must be declared previously
430038  // the type 'inode_t' as made of a type 'struct inode', then the
430039  // structure 'inode' can be described. But for a matter of coherence,
430040  // all other structured data declared inside this file follow the
430041  // same procedure.
430042  //
430043  typedef struct sb        sb_t;
430044  typedef struct inode     inode_t;
430045  typedef struct file      file_t;
430046  typedef struct fd        fd_t;
430047  typedef struct directory directory_t;
430048  //------------------------------------------------------------------------
430049  #define SB_MAX_SLOTS  2                 // Handle max 2 file systems.
430050
430051  struct sb {                             // File system super block:
430052      uint16_t      inodes;               // inodes available;
430053      uint16_t      zones;                // zones available (disk size);
430054      uint16_t      map_inode_blocks;     // inode bit map blocks;
430055      uint16_t      map_zone_blocks;      // data-zone bit map blocks;
430056      uint16_t      first_data_zone;      // first data-zone;
430057      uint16_t      log2_size_zone;       // log_2 (size_zone/block_size);
430058      uint32_t      max_file_size;        // max file size in bytes;
430059      uint16_t      magic_number;         // file system magic number.
430060      //--------------------------------------------------------------
430061      // Extra management data, not saved inside the file system
430062      // super block.
430063      //--------------------------------------------------------------
430064      dev_t         device;                       // FS device [3]
430065      inode_t      *inode_mounted_on;             // [4]
430066      blksize_t     blksize;                      // Calculated zone size.
430067      int           options;                      // [5]
430068      uint16_t      map_inode[SB_MAP_INODE_SIZE];
430069      uint16_t      map_zone[SB_MAP_ZONE_SIZE];
430070      char          changed;
430071  };
430072
430073  extern sb_t    sb_table[SB_MAX_SLOTS];
430074  //
430075  // [3] the member 'device' must be kept at the same position, because
430076  //     it is used to calculate the super block header size, saved on
430077  //     disk.
430078  //
430079  // [4] If this pointer is not NULL, the super block is related to a
430080  //     device mounted on a directory. The inode of such directory is
430081  //     recorded here. Please note that it is type 'void *', instead of
430082  //     type 'inode_t', because type 'inode_t' is declared after type
430083  //     'sb_t'.
430084  //     Please note that the type 'sb_t' is declared before the
430085  //     type 'inode_t', but this member points to a type 'inode_t'.
430086  //     This is the reason because it was necessary to declare first
430087  //     the type 'inode_t' as made of 'struct inode', to be described
430088  //     later. For coherence, all derived type made of structured data,
430089  //     are first declared as structure, and then, later, described.
430090  //
430091  // [5] Mount options can be only 'MOUNT_DEFAULT' or 'MOUNT_RO',
430092  //     as defined inside file 'lib/sys/os16.h'.
430093  //
430094  //------------------------------------------------------------------------
430095  #define INODE_MAX_SLOTS           (8 * OPEN_MAX)
430096
430097  struct inode {                          // Inode (32 byte total):
430098      mode_t        mode;                 // file type and permissions;
```



```
430099      uid_t         uid;                  // user ID (16 bit);
430100      ssize_t       size;                 // file size in bytes;
430101      time_t        time;                 // file data modification time;
430102      uint8_t       gid;                  // group ID (8 bit);
430103      uint8_t       links;                // links to the inode;
430104      zno_t         direct[7];            // direct zones;
430105      zno_t         indirect1;            // indirect zones;
430106      zno_t         indirect2;            // double indirect zones.
430107      //--------------------------------------------------------------
430108      // Extra management data, not saved inside the disk file system.
430109      //--------------------------------------------------------------
430110      sb_t         *sb;                   // Inode's super block. [7]
430111      ino_t         ino;                  // Inode number.
430112      sb_t         *sb_attached;          // [8]
430113      blkcnt_t      blkcnt;               // Rounded size/blksize.
430114      unsigned char references;           // Run time active references.
430115      char          changed;              // 1 == to be saved.
430116  };
430117
430118  extern inode_t inode_table[INODE_MAX_SLOTS];
430119  //
430120  // [7] the member 'sb' must be kept at the same position, because
430121  //     it is used to calculate the inode header size, saved on disk.
430122  //
430123  // [8] If the inode is a mount point for another device, the other
430124  //     super block pointer is saved inside 'sb_attached'.
430125  //
430126  //------------------------------------------------------------------------
430127  #define FILE_MAX_SLOTS             (16 * OPEN_MAX)
430128
430129  struct file {
430130      int           references;
430131      off_t         offset;               // File position.
430132      int           oflags;               // Open mode: r/w/r+w [9]
430133      inode_t      *inode;
430134  };
430135
430136  extern file_t file_table[FILE_MAX_SLOTS];
430137  //
430138  // [9] the member 'oflags' can get only O_RDONLY, O_WRONLY, O_RDWR,
430139  //     (from header 'fcntl.h') combined with OR binary operator.
430140  //
430141  //------------------------------------------------------------------------
430142  struct fd {
430143      int           fl_flags;             // File status flags and file
430144                                          // access modes. [10]
430145      int           fd_flags;             // File descriptor flags:
430146                                          // currently only FD_CLOEXEC.
430147      file_t       *file;                 // Pointer to the file table.
430148  };
430149  //
430150  // [10] the member 'fl_flags' can get only O_RDONLY, O_WRONLY, O_RDWR,
430151  //      O_CREAT, O_EXCL, O_NOCTTY, O_TRUNC and O_APPEND
430152  //      (from header 'fcntl.h') combined with OR binary
430153  //      operator. Options like O_DSYNC, O_NONBLOCK, O_RSYNC and O_SYNC
430154  //      are not taken into consideration by os16.
430155  //
430156  //------------------------------------------------------------------------
430157  struct directory {                      // Directory entry:
430158      ino_t         ino;                  // inode number;
430159      char          name[NAME_MAX];       // file name.
430160  };
430161  //------------------------------------------------------------------------
430162  sb_t    *sb_reference    (dev_t device);
430163  sb_t    *sb_mount        (dev_t device, inode_t **inode_mnt,
430164                            int options);
430165  sb_t    *sb_get          (dev_t device, sb_t *sb);
430166  int      sb_save         (sb_t *sb);
430167  int      sb_zone_status  (sb_t *sb, zno_t zone);
430168  int      sb_inode_status (sb_t *sb, ino_t ino);
430169  //------------------------------------------------------------------------
430170  zno_t    zone_alloc           (sb_t *sb);
430171  int      zone_free            (sb_t *sb, zno_t zone);
430172  int      zone_read            (sb_t *sb, zno_t zone, void *buffer);
430173  int      zone_write           (sb_t *sb, zno_t zone, void *buffer);
430174  //------------------------------------------------------------------------
430175  inode_t *inode_reference      (dev_t device, ino_t ino);
430176  inode_t *inode_alloc          (dev_t device, mode_t mode, uid_t uid);
430177  int      inode_free           (inode_t *inode);
430178  inode_t *inode_get            (dev_t device, ino_t ino);
430179  int      inode_save           (inode_t *inode);
430180  int      inode_put            (inode_t *inode);
430181  int      inode_truncate       (inode_t *inode);
430182  zno_t    inode_zone           (inode_t *inode, zno_t fzone, int write);
430183  inode_t *inode_stdio_dev_make (dev_t device, mode_t mode);
430184  blkcnt_t inode_fzones_read    (inode_t *inode, zno_t zone_start,
430185                                 void *buffer, blkcnt_t blkcnt);
430186  blkcnt_t inode_fzones_write   (inode_t *inode, zno_t zone_start,
430187                                 void *buffer, blkcnt_t blkcnt);
430188  ssize_t  inode_file_read      (inode_t *inode, off_t offset,
430189                                 void *buffer, size_t count, int *eof);
430190  ssize_t  inode_file_write     (inode_t *inode, off_t offset,
430191                                 void *buffer, size_t count);
430192  int      inode_check          (inode_t *inode, mode_t type,
430193                                 int perm, uid_t uid);
430194  int      inode_dir_empty      (inode_t *inode);
430195  //------------------------------------------------------------------------
430196  file_t  *file_reference       (int fno);
430197  file_t  *file_stdio_dev_make  (dev_t device, mode_t mode, int oflags);
430198  //------------------------------------------------------------------------
430199  inode_t *path_inode      (pid_t pid, const char *path);
```

```
430200 int      path_chdir      (pid_t pid, const char *path);
430201 dev_t    path_device     (pid_t pid, const char *path);
430202 int      path_full       (const char *path,
430203                           const char *path_cwd,
430204                           char *full_path);
430205 int      path_fix        (char *path);
430206 inode_t *path_inode_link (pid_t pid, const char *path, inode_t *inode,
430207                           mode_t mode);
430208 int      path_link       (pid_t pid, const char *path_old,
430209                           const char *path_new);
430210 int      path_mkdir      (pid_t pid, const char *path, mode_t mode);
430211 int      path_mknod      (pid_t pid, const char *path, mode_t mode,
430212                           dev_t device);
430213 int      path_mount      (pid_t pid, const char *path_dev,
430214                           const char *path_mnt,
430215                           int options);
430216 int      path_umount     (pid_t pid, const char *path_mnt);
430217 int      path_stat       (pid_t pid, const char *path,
430218                           struct stat *buffer);
430219 int      path_chmod      (pid_t pid, const char *path, mode_t mode);
430220 int      path_chown      (pid_t pid, const char *path, uid_t uid,
430221                           gid_t gid);
430222 int      path_unlink     (pid_t pid, const char *path);
430223 //-------------------------------------------------------------------------
430224 fd_t    *fd_reference (pid_t pid, int *fdn);
430225 int      fd_chmod     (pid_t pid, int fdn, mode_t mode);
430226 int      fd_chown     (pid_t pid, int fdn, uid_t uid, gid_t gid);
430227 int      fd_close     (pid_t pid, int fdn);
430228 int      fd_fcntl     (pid_t pid, int fdn, int cmd, int arg);
430229 int      fd_dup       (pid_t pid, int fdn_old, int fdn_min);
430230 int      fd_dup2      (pid_t pid, int fdn_old, int fdn_new);
430231 off_t    fd_lseek     (pid_t pid, int fdn, off_t offset, int whence);
430232 int      fd_open      (pid_t pid, const char *path, int oflags,
430233                        mode_t mode);
430234 ssize_t  fd_read      (pid_t pid, int fdn, void *buffer, size_t count,
430235                        int *eof);
430236 int      fd_stat      (pid_t pid, int fdn, struct stat *buffer);
430237 ssize_t  fd_write     (pid_t pid, int fdn, const void *buffer,
430238                        size_t count);
430239 //-------------------------------------------------------------------------
430240
430241 #endif
```

kernel/fs/fd_chmod.c

«

Si veda la sezione i159.3.1.

```
440001 #include <kernel/proc.h>
440002 #include <kernel/k_libc.h>
440003 #include <sys/stat.h>
440004 #include <errno.h>
440005 //-------------------------------------------------------------------------
440006 int
440007 fd_chmod (pid_t pid, int fdn, mode_t mode)
440008 {
440009     proc_t   *ps;
440010     inode_t  *inode;
440011     //
440012     // Get process.
440013     //
440014     ps = proc_reference (pid);
440015     //
440016     // Verify if the file descriptor is valid.
440017     //
440018     if (ps->fd[fdn].file == NULL)
440019       {
440020         errset (EBADF);         // Bad file descriptor.
440021         return (-1);
440022       }
440023     //
440024     // Reach the inode.
440025     //
440026     inode = ps->fd[fdn].file->inode;
440027     //
440028     // Verify to be the owner, or at least to be UID == 0.
440029     //
440030     if (ps->euid != inode->uid && ps->euid != 0)
440031       {
440032         errset (EACCES);        // Permission denied.
440033         return (-1);
440034       }
440035     //
440036     // Update the mode: the file type is kept and the
440037     // rest is taken form the parameter 'mode'.
440038     //
440039     inode->mode = (S_IFMT & inode->mode) | (~S_IFMT & mode);
440040     //
440041     // Save the inode.
440042     //
440043     inode->changed = 1;
440044     inode_save (inode);
440045     //
440046     // Return.
440047     //
440048     return (0);
440049 }
```


kernel/fs/fd_chown.c

«

Si veda la sezione i159.3.2.

```
450001 #include <kernel/proc.h>
450002 #include <kernel/k_libc.h>
450003 #include <errno.h>
450004 //-------------------------------------------------------------------------
450005 int
450006 fd_chown (pid_t pid, int fdn, uid_t uid, gid_t gid)
450007 {
450008     proc_t   *ps;
450009     inode_t  *inode;
450010     //
450011     // Get process.
450012     //
450013     ps = proc_reference (pid);
450014     //
450015     // Verify if the file descriptor is valid.
450016     //
450017     if (ps->fd[fdn].file == NULL)
450018       {
450019         errset (EBADF);         // Bad file descriptor.
450020         return (-1);
450021       }
450022     //
450023     // Reach the inode.
450024     //
450025     inode = ps->fd[fdn].file->inode;
450026     //
450027     // Verify to be root, as the ability to change group
450028     // is not taken into consideration.
450029     //
450030     if (ps->euid != 0)
450031       {
450032         errset (EACCES);        // Permission denied.
450033         return (-1);
450034       }
450035     //
450036     // Update the ownership.
450037     //
450038     if (uid != -1)
450039       {
450040         inode->uid     = uid;
450041         inode->changed = 1;
450042       }
450043     if (gid != -1)
450044       {
450045         inode->gid     = gid;
450046         inode->changed = 1;
450047       }
450048     //
450049     // Save the inode.
450050     //
450051     inode->changed = 1;
450052     inode_save (inode);
450053     //
450054     // Return.
450055     //
450056     return (0);
450057 }
```

kernel/fs/fd_close.c

«

Si veda la sezione i159.3.3.

```
460001 #include <kernel/proc.h>
460002 #include <kernel/k_libc.h>
460003 #include <errno.h>
460004 //-------------------------------------------------------------------------
460005 int
460006 fd_close (pid_t pid, int fdn)
460007 {
460008     inode_t           *inode;
460009     file_t            *file;
460010     fd_t *fd;
460011     //
460012     // Get file descriptor.
460013     //
460014     fd = fd_reference (pid, &fdn);
460015     if (fd == NULL                ||
460016         fd->file == NULL          ||
460017         fd->file->inode == NULL )
460018       {
460019         errset (EBADF);         // Bad file descriptor.
460020         return (-1);
460021       }
460022     //
460023     // Get file.
460024     //
460025     file = fd->file;
460026     //
460027     // Get inode.
460028     //
460029     inode = file->inode;
460030     //
460031     // Reduce references inside the file table item
460032     // and remove item if it reaches zero.
460033     //
460034     file->references--;
460035     if (file->references == 0)
```

```
460036        {
460037          file->oflags = 0;
460038          file->inode  = NULL;
460039          //
460040          // Put inode, because there are no more file references.
460041          //
460042          inode_put (inode);
460043        }
460044      //
460045      // Remove file descriptor.
460046      //
460047      fd->fl_flags = 0;
460048      fd->fd_flags = 0;
460049      fd->file  = NULL;
460050      //
460051      //
460052      //
460053      return (0);
460054  }
```

kernel/fs/fd_dup.c

«

Si veda la sezione i159.3.4.

```
470001  #include <kernel/proc.h>
470002  #include <kernel/k_libc.h>
470003  #include <errno.h>
470004  #include <fcntl.h>
470005  //-------------------------------------------------------------------------
470006  int
470007  fd_dup (pid_t pid, int fdn_old, int fdn_min)
470008  {
470009      proc_t *ps;
470010      int     fdn_new;
470011      //
470012      // Verify argument.
470013      //
470014      if (fdn_min < 0 || fdn_min >= OPEN_MAX)
470015        {
470016          errset (EINVAL);                 // Invalid argument.
470017          return (-1);
470018        }
470019      //
470020      // Get process.
470021      //
470022      ps = proc_reference (pid);
470023      //
470024      // Verify if 'fdn_old' is a valid value.
470025      //
470026      if (fdn_old < 0                     ||
470027          fdn_old >= OPEN_MAX             ||
470028          ps->fd[fdn_old].file == NULL)
470029        {
470030          errset (EBADF);                 // Bad file descriptor.
470031          return (-1);
470032        }
470033      //
470034      // Find the first free slot and duplicate the file descriptor.
470035      //
470036      for (fdn_new = fdn_min; fdn_new < OPEN_MAX; fdn_new++)
470037        {
470038          if (ps->fd[fdn_new].file == NULL)
470039            {
470040              ps->fd[fdn_new].fl_flags = ps->fd[fdn_old].fl_flags;
470041              ps->fd[fdn_new].fd_flags
470042                  = ps->fd[fdn_old].fd_flags & ~FD_CLOEXEC;
470043              ps->fd[fdn_new].file = ps->fd[fdn_old].file;
470044              ps->fd[fdn_new].file->references++;
470045              return (fdn_new);
470046            }
470047        }
470048      //
470049      // No fd slot available.
470050      //
470051      errset (EMFILE);            // Too many open files.
470052      return (-1);
470053  }
```

kernel/fs/fd_dup2.c

«

Si veda la sezione i159.3.4.

```
480001  #include <kernel/proc.h>
480002  #include <kernel/k_libc.h>
480003  #include <errno.h>
480004  #include <fcntl.h>
480005  //-------------------------------------------------------------------------
480006  int
480007  fd_dup2 (pid_t pid, int fdn_old, int fdn_new)
480008  {
480009      proc_t *ps;
480010      int     status;
480011      //
480012      // Get process.
480013      //
480014      ps = proc_reference (pid);
480015      //
480016      // Verify if 'fdn_old' is a valid value.
480017      //
480018      if (fdn_old < 0                     ||
```



```
480019          fdn_old >= OPEN_MAX             ||
480020          ps->fd[fdn_old].file == NULL)
480021        {
480022          errset (EBADF);                 // Bad file descriptor.
480023          return (-1);
480024        }
480025      //
480026      // Check if 'fd_old' and 'fd_new' are the same.
480027      //
480028      if (fdn_old == fdn_new)
480029        {
480030          return (fdn_new);
480031        }
480032      //
480033      // Close 'fdn_new' if it is open and copy 'fdn_old' into it.
480034      //
480035      if (ps->fd[fdn_new].file != NULL)
480036        {
480037          status = fd_close (pid, fdn_new);
480038          if (status != 0)
480039            {
480040              return (-1);
480041            }
480042        }
480043      ps->fd[fdn_new].fl_flags = ps->fd[fdn_old].fl_flags;
480044      ps->fd[fdn_new].fd_flags = ps->fd[fdn_old].fd_flags & ~FD_CLOEXEC;
480045      ps->fd[fdn_new].file = ps->fd[fdn_old].file;
480046      ps->fd[fdn_new].file->references++;
480047      return (fdn_new);
480048  }
```

kernel/fs/fd_fcntl.c

«

Si veda la sezione i159.3.6.

```
490001  #include <kernel/proc.h>
490002  #include <kernel/k_libc.h>
490003  #include <errno.h>
490004  #include <fcntl.h>
490005  //-------------------------------------------------------------------------
490006  int
490007  fd_fcntl (pid_t pid, int fdn, int cmd, int arg)
490008  {
490009      proc_t   *ps;
490010      inode_t  *inode;
490011      int       mask;
490012      //
490013      // Get process.
490014      //
490015      ps = proc_reference (pid);
490016      //
490017      // Verify if the file descriptor is valid.
490018      //
490019      if (ps->fd[fdn].file == NULL)
490020        {
490021          errset (EBADF);                 // Bad file descriptor.
490022          return (-1);
490023        }
490024      //
490025      // Reach the inode.
490026      //
490027      inode = ps->fd[fdn].file->inode;
490028      //
490029      //
490030      //
490031      switch (cmd)
490032        {
490033          case F_DUPFD:
490034              return (fd_dup (pid, fdn, arg));
490035              break;
490036          case F_GETFD:
490037              return (ps->fd[fdn].fd_flags);
490038              break;
490039          case F_SETFD:
490040              ps->fd[fdn].fd_flags = arg;
490041              return (0);
490042          case F_GETFL:
490043              return (ps->fd[fdn].fl_flags);
490044          case F_SETFL:
490045              //
490046              // Calculate a mask with bits that are not to be set.
490047              //
490048              mask = (O_ACCMODE
490049                    | O_CREAT
490050                    | O_EXCL
490051                    | O_NOCTTY
490052                    | O_TRUNC);
490053              //
490054              // Set to zero the bits that are not to be set from
490055              // the argument.
490056              //
490057              arg = (arg & ~mask);
490058              //
490059              // Set to zero the bit that *are* to be set.
490060              //
490061              ps->fd[fdn].fl_flags &= mask;
490062              //
490063              // Set the bits, already filtered inside the argument.
490064              //
490065              ps->fd[fdn].fl_flags |= arg;
490066              //
```

```
490067              return (0);
490068          default:
490069              errset (EINVAL);                    // Not implemented.
490070              return (-1);
490071          }
490072  }
```

kernel/fs/fd_lseek.c

«

Si veda la sezione i159.3.7.

```
500001  #include <kernel/proc.h>
500002  #include <kernel/k_libc.h>
500003  #include <errno.h>
500004  //---------------------------------------------------------------------
500005  off_t
500006  fd_lseek (pid_t pid, int fdn, off_t offset, int whence)
500007  {
500008      inode_t           *inode;
500009      file_t            *file;
500010      fd_t *fd;
500011      off_t              test_offset;
500012      //
500013      // Get file descriptor.
500014      //
500015      fd = fd_reference (pid, &fdn);
500016      if (fd == NULL               ||
500017          fd->file == NULL         ||
500018          fd->file->inode == NULL )
500019        {
500020          errset (EBADF);                 // Bad file descriptor.
500021          return (-1);
500022        }
500023      //
500024      // Get file table item.
500025      //
500026      file = fd->file;
500027      //
500028      // Get inode.
500029      //
500030      inode = file->inode;
500031      //
500032      // Change position depending on the 'whence' parameter.
500033      //
500034      if (whence == SEEK_SET)
500035        {
500036          if (offset < 0)
500037            {
500038              errset (EINVAL);                    // Invalid argument.
500039              return ((off_t) -1);
500040            }
500041          else
500042            {
500043              fd->file->offset = offset;
500044            }
500045        }
500046      else if (whence == SEEK_CUR)
500047        {
500048          test_offset  = fd->file->offset;
500049          test_offset += offset;
500050          if (test_offset < 0)
500051            {
500052              errset (EINVAL);                    // Invalid argument.
500053              return ((off_t) -1);
500054            }
500055          else
500056            {
500057              fd->file->offset = test_offset;
500058            }
500059        }
500060      else if (whence == SEEK_END)
500061        {
500062          test_offset  = inode->size;
500063          test_offset += offset;
500064          if (test_offset < 0)
500065            {
500066              errset (EINVAL);                    // Invalid argument.
500067              return ((off_t) -1);
500068            }
500069          else
500070            {
500071              fd->file->offset = test_offset;
500072            }
500073        }
500074      else
500075        {
500076          errset (EINVAL);                        // Invalid argument.
500077          return ((off_t) -1);
500078        }
500079      //
500080      // Return the new file position.
500081      //
500082      return (fd->file->offset);
500083  }
```


kernel/fs/fd_open.c

«

Si veda la sezione i159.3.8.

```
510001  #include <kernel/proc.h>
510002  #include <kernel/k_libc.h>
510003  #include <errno.h>
510004  #include <fcntl.h>
510005  //---------------------------------------------------------------------
510006  int
510007  fd_open (pid_t pid, const char *path, int oflags, mode_t mode)
510008  {
510009      proc_t   *ps;
510010      inode_t  *inode;
510011      int       status;
510012      file_t   *file;
510013      fd_t     *fd;
510014      int       fdn;
510015      char      full_path[PATH_MAX];
510016      int       perm;
510017      tty_t    *tty;
510018      mode_t    umask;
510019      int       errno_save;
510020      //
510021      // Get process.
510022      //
510023      ps = proc_reference (pid);
510024      //
510025      // Correct the mode with the umask. As it is not a directory, to the
510026      // mode are removed execution and sticky permissions.
510027      //
510028      umask = ps->umask | 01111;
510029      mode &= ~umask;
510030      //
510031      // Check open options.
510032      //
510033      if (oflags & O_WRONLY)
510034        {
510035          //
510036          // The file is to be opened for write, or for read/write.
510037          // Try to get inode.
510038          //
510039          inode = path_inode (pid, path);
510040          if (inode == NULL)
510041            {
510042              //
510043              // Cannot get the inode. See if there is the creation
510044              // option.
510045              //
510046              if (oflags & O_CREAT)
510047                {
510048                  //
510049                  // Try to create the missing inode: the file must be a
510050                  // regular one, so add the mode.
510051                  //
510052                  path_full (path, ps->path_cwd, full_path);
510053                  inode = path_inode_link (pid, full_path, NULL,
510054                                           (mode | S_IFREG));
510055                  if (inode == NULL)
510056                    {
510057                      //
510058                      // Sorry: cannot create the inode! Variable 'errno'
510059                      // is already set by 'path_inode_link()'.
510060                      //
510061                      errset (errno);
510062                      return (-1);
510063                    }
510064                }
510065              else
510066                {
510067                  //
510068                  // Cannot open the inode. Variable 'errno'
510069                  // should be already set by 'path_inode()'.
510070                  //
510071                  errset (errno);
510072                  return (-1);
510073                }
510074            }
510075          //
510076          // The inode was read or created: check if it must be
510077          // truncated. It can be truncated only if it is a regular
510078          // file.
510079          //
510080          if (oflags & O_TRUNC && inode->mode & S_IFREG)
510081            {
510082              //
510083              // Truncate inode.
510084              //
510085              status = inode_truncate (inode);
510086              if (status != 0)
510087                {
510088                  //
510089                  // Cannot truncate the inode: release it and return.
510090                  // But this error should never happen, because the
510091                  // function 'inode_truncate()' will not return any
510092                  // other value than zero.
510093                  //
510094                  errno_save = errno;
510095                  inode_put (inode);
510096                  errset (errno_save);
510097                  return (-1);
510098                }
```

```
510099            }
510100        }
510101      else
510102        {
510103          //
510104          // The file is to be opened for read, but not for write.
510105          // Try to get inode.
510106          //
510107          inode = path_inode (pid, path);
510108          if (inode == NULL)
510109            {
510110              //
510111              // Cannot open the file.
510112              //
510113              errset (errno);
510114              return (-1);
510115            }
510116        }
510117      //
510118      // An inode was opened: check type and access permissions.
510119      // All file types are good, even directories, as the type
510120      // DIR is implemented through file descriptors.
510121      //
510122      perm = 0;
510123      if (oflags & O_RDONLY) perm |= 4;
510124      if (oflags & O_WRONLY) perm |= 2;
510125      status = inode_check (inode, S_IFMT, perm, ps->uid);
510126      if (status != 0)
510127        {
510128          //
510129          // The file type is not correct or the user does not have
510130          // permissions.
510131          //
510132          return (-1);
510133        }
510134      //
510135      // Allocate the file, inside the file table.
510136      //
510137      file = file_reference (-1);
510138      if (file == NULL)
510139        {
510140          //
510141          // Cannot allocate the file inside the file table: release the
510142          // inode, update 'errno' and return.
510143          //
510144          inode_put (inode);
510145          errset (ENFILE);        // Too many files open in system.
510146          return (-1);
510147        }
510148      //
510149      // Put some data inside the file item. Only options
510150      // O_RDONLY and O_WRONLY are kept here, because the O_APPEND
510151      // is saved inside the file descriptor table.
510152      //
510153      file->references = 1;
510154      file->oflags     = (oflags & (O_RDONLY | O_WRONLY));
510155      file->inode      = inode;
510156      //
510157      // Allocate the file descriptor: variable 'fdn' will be modified
510158      // by the call to 'fd_reference()'.
510159      //
510160      fdn = -1;
510161      fd = fd_reference (pid, &fdn);
510162      if (fd == NULL)
510163        {
510164          //
510165          // Cannot allocate the file descriptor: remove the item from
510166          // file table.
510167          //
510168          file->references = 0;
510169          file->oflags     = 0;
510170          file->inode      = NULL;
510171          //
510172          // Release the inode.
510173          //
510174          inode_put (inode);
510175          //
510176          // Return an error.
510177          //
510178          errset (EMFILE);        // Too many open files.
510179          return (-1);
510180        }
510181      //
510182      // File descriptor allocated: put some data inside the
510183      // file descriptor item.
510184      //
510185      fd->fl_flags = (oflags & (O_RDONLY | O_WRONLY | O_APPEND));
510186      fd->fd_flags = 0;
510187      fd->file = file;
510188      fd->file->offset = 0;
510189      //
510190      // Check if it is a terminal (currently only consoles), if it is
510191      // opened for read and write, and if it have to be set as the
510192      // controlling terminal. This thing is done here because there is
510193      // not a real device driver.
510194      //
510195      if ((S_ISCHR (inode->mode)) &&
510196          (oflags & O_RDONLY)     &&
510197          (oflags & O_WRONLY))
510198        {
510199          //
```



```
510200          // The inode is a character special file (related to a character
510201          // device), opened for read and write!
510202          //
510203          if ((inode->direct[0] & 0xFF00) == (DEV_CONSOLE_MAJOR << 8))
510204            {
510205              //
510206              // It is a terminal (currently only consoles are possible).
510207              // Get the tty reference.
510208              //
510209              tty = tty_reference ((dev_t) inode->direct[0]);
510210              //
510211              // Verify that the terminal is not already the controlling
510212              // terminal of some process group.
510213              //
510214              if (tty->pgrp == 0)
510215                {
510216                  //
510217                  // The terminal is free: verify if the current process
510218                  // needs a controlling terminal.
510219                  //
510220                  if (ps->device_tty == 0 && ps->pgrp == pid)
510221                    {
510222                      //
510223                      // It is a group leader with no controlling
510224                      // terminal: set the controlling terminal.
510225                      //
510226                      ps->device_tty = inode->direct[0];
510227                      tty->pgrp      = ps->pgrp;
510228                    }
510229                }
510230            }
510231        }
510232      //
510233      // Return the file descriptor.
510234      //
510235      return (fdn);
510236  }
```

kernel/fs/fd_read.c

«

Si veda la sezione i159.3.9.

```
520001  #include <kernel/proc.h>
520002  #include <kernel/k_libc.h>
520003  #include <errno.h>
520004  #include <fcntl.h>
520005  //---------------------------------------------------------------------
520006  ssize_t
520007  fd_read (pid_t pid, int fdn, void *buffer, size_t count, int *eof)
520008  {
520009      fd_t  *fd;
520010      ssize_t  size_read;
520011      //
520012      // Get file descriptor.
520013      //
520014      fd = fd_reference (pid, &fdn);
520015      if (fd              == NULL   ||
520016          fd->file        == NULL   ||
520017          fd->file->inode == NULL )
520018        {
520019          errset (EBADF);                 // Bad file descriptor.
520020          return ((ssize_t) -1);
520021        }
520022      //
520023      // Check if it is opened for read.
520024      //
520025      if (!(fd->file->oflags & O_RDONLY))
520026        {
520027          //
520028          // The file is not opened for read.
520029          //
520030          errset (EINVAL);                // Invalid argument.
520031          return ((ssize_t) -1);
520032        }
520033      //
520034      // It is not a mistake to read a directory, as 'dirent.h' is
520035      // implemented through file descriptors.
520036      //
520037      //  //
520038      //  // Check if it is a directory.
520039      //  //
520040      //  if (fd->file->inode->mode & S_IFDIR)
520041      //    {
520042      //      errset (EISDIR);                // Is a directory.
520043      //      return ((ssize_t) -1);
520044      //    }
520045      //
520046      // Check the kind of file to be read and read it.
520047      //
520048      if  (S_ISBLK (fd->file->inode->mode)
520049        || S_ISCHR (fd->file->inode->mode))
520050        {
520051          //
520052          // A device is to be read.
520053          //
520054          size_read = dev_io (pid, (dev_t) fd->file->inode->direct[0],
520055                              DEV_READ, fd->file->offset, buffer, count,
520056                              eof);
520057        }
520058      else if (S_ISREG (fd->file->inode->mode))
520059        {
```

```
520060          //
520061          // A regular file is to be read.
520062          //
520063          size_read = inode_file_read (fd->file->inode, fd->file->offset,
520064                                       buffer, count, eof);
520065        }
520066      else if (S_ISDIR (fd->file->inode->mode))
520067        {
520068          //
520069          // A directory, is to be read.
520070          //
520071          size_read = inode_file_read (fd->file->inode, fd->file->offset,
520072                                       buffer, count, eof);
520073        }
520074      else
520075        {
520076          //
520077          // Unsupported file type.
520078          //
520079          errset (E_FILE_TYPE_UNSUPPORTED);       //File type unsupported.
520080          return ((ssize_t) -1);
520081        }
520082      //
520083      // Update the file descriptor internal offset.
520084      //
520085      if (size_read > 0)
520086        {
520087          fd->file->offset += size_read;
520088        }
520089      //
520090      // Just return the size read, even if it is an error. Please note
520091      // that a size of zero might tell that it is the end of file, or
520092      // just that the read should be retried.
520093      //
520094      return (size_read);
520095  }
```

kernel/fs/fd_reference.c

«

Si veda la sezione i159.3.10.

```
530001  #include <kernel/proc.h>
530002  #include <kernel/k_libc.h>
530003  #include <errno.h>
530004  //-------------------------------------------------------------------------
530005  fd_t *
530006  fd_reference (pid_t pid, int *fdn)
530007  {
530008      proc_t *ps;
530009      //
530010      // Get process.
530011      //
530012      ps = proc_reference (pid);
530013      //
530014      // See what to do.
530015      //
530016      if (*fdn < 0)
530017        {
530018          //
530019          // Find the first free slot.
530020          //
530021          for (*fdn = 0; *fdn < OPEN_MAX; (*fdn)++)
530022            {
530023              if (ps->fd[*fdn].file == NULL)
530024                {
530025                  return (&(ps->fd[*fdn]));
530026                }
530027            }
530028          *fdn = -1;
530029          return (NULL);
530030        }
530031      else
530032        {
530033          if (*fdn < OPEN_MAX)
530034            {
530035              //
530036              // Might return even a free file descriptor.
530037              //
530038              return (&(ps->fd[*fdn]));
530039            }
530040          else
530041            {
530042              return (NULL);
530043            }
530044        }
530045  }
```

kernel/fs/fd_stat.c

«

Si veda la sezione i159.3.50.

```
540001  #include <kernel/proc.h>
540002  #include <kernel/k_libc.h>
540003  #include <errno.h>
540004  #include <fcntl.h>
540005  //-------------------------------------------------------------------------
540006  int
540007  fd_stat (pid_t pid, int fdn, struct stat *buffer)
540008  {
540009      proc_t   *ps;
```



```
540010      inode_t  *inode;
540011      //
540012      // Get process.
540013      //
540014      ps = proc_reference (pid);
540015      //
540016      // Verify if the file descriptor is valid.
540017      //
540018      if (ps->fd[fdn].file == NULL)
540019        {
540020          errset (EBADF);                 // Bad file descriptor.
540021          return (-1);
540022        }
540023      //
540024      // Reach the inode.
540025      //
540026      inode = ps->fd[fdn].file->inode;
540027      //
540028      // Inode loaded: update the buffer.
540029      //
540030      buffer->st_dev      = inode->sb->device;
540031      buffer->st_ino      = inode->ino;
540032      buffer->st_mode     = inode->mode;
540033      buffer->st_nlink    = inode->links;
540034      buffer->st_uid      = inode->uid;
540035      buffer->st_gid      = inode->gid;
540036      if (S_ISBLK (buffer->st_mode) || S_ISCHR (buffer->st_mode))
540037        {
540038          buffer->st_rdev = inode->direct[0];
540039        }
540040      else
540041        {
540042          buffer->st_rdev = 0;
540043        }
540044      buffer->st_size     = inode->size;
540045      buffer->st_atime    = inode->time;  // All times are the same for
540046      buffer->st_mtime    = inode->time;  // Minix 1 file system.
540047      buffer->st_ctime    = inode->time;  //
540048      buffer->st_blksize  = inode->sb->blksize;
540049      buffer->st_blocks   = inode->blkcnt;
540050      //
540051      // If the inode is a device special file, the 'st_rdev' value is
540052      // taken from the first direct zone (as of Minix 1 organization).
540053      //
540054      if (S_ISBLK(inode->mode) || S_ISCHR(inode->mode))
540055        {
540056          buffer->st_rdev = inode->direct[0];
540057        }
540058      else
540059        {
540060          buffer->st_rdev = 0;
540061        }
540062      //
540063      // Return.
540064      //
540065      return (0);
540066  }
```

kernel/fs/fd_write.c

«

Si veda la sezione i159.3.12.

```
550001  #include <kernel/proc.h>
550002  #include <kernel/k_libc.h>
550003  #include <errno.h>
550004  #include <fcntl.h>
550005  //-------------------------------------------------------------------------
550006  ssize_t
550007  fd_write (pid_t pid, int fdn, const void *buffer, size_t count)
550008  {
550009      proc_t  *ps;
550010      fd_t    *fd;
550011      ssize_t  size_written;
550012      //
550013      // Get process.
550014      //
550015      ps = proc_reference (pid);
550016      //
550017      // Get file descriptor.
550018      //
550019      fd = fd_reference (pid, &fdn);
550020      if (fd              == NULL   ||
550021          fd->file        == NULL   ||
550022          fd->file->inode == NULL )
550023        {
550024          //
550025          // The file descriptor pointer is not valid.
550026          //
550027          errset (EBADF);                 // Bad file descriptor.
550028          return ((ssize_t) -1);
550029        }
550030      //
550031      // Check if it is opened for write.
550032      //
550033      if (!(fd->file->oflags & O_WRONLY))
550034        {
550035          //
550036          // The file is not opened for write.
550037          //
550038          errset (EINVAL);                // Invalid argument.
550039          return ((ssize_t) -1);
```

```
550040        }
550041      //
550042      // Check if it is a directory.
550043      //
550044      if (fd->file->inode->mode & S_IFDIR)
550045        {
550046          errset (EISDIR);                // Is a directory.
550047          return ((ssize_t) -1);
550048        }
550049      //
550050      // It should be a valid type of file to be written. Check if it is
550051      // opened in append mode: if so, must move the write offset to the
550052      // end.
550053      //
550054      if (fd->fl_flags & O_APPEND)
550055        {
550056          fd->file->offset = fd->file->inode->size;
550057        }
550058      //
550059      // Check the kind of file to be written and write it.
550060      //
550061      if (fd->file->inode->mode & S_IFBLK ||
550062          fd->file->inode->mode & S_IFCHR)
550063        {
550064          //
550065          // A device is to be written.
550066          //
550067          size_written = dev_io (pid, (dev_t) fd->file->inode->direct[0],
550068                                 DEV_WRITE, fd->file->offset, buffer,
550069                                 count, NULL);
550070        }
550071      else if (fd->file->inode->mode & S_IFREG)
550072        {
550073          //
550074          // A regular file is to be written.
550075          //
550076          size_written = inode_file_write (fd->file->inode,
550077                                           fd->file->offset,
550078                                           buffer, count);
550079        }
550080      else
550081        {
550082          //
550083          // Unsupported file type.
550084          //
550085          errset (E_FILE_TYPE_UNSUPPORTED);       //File type unsupported.
550086          return ((ssize_t) -1);
550087        }
550088      //
550089      // Update the file descriptor internal offset.
550090      //
550091      if (size_written > 0)
550092        {
550093          fd->file->offset += size_written;
550094        }
550095      //
550096      // Just return the size written, even if it is an error.
550097      //
550098      return (size_written);
550099  }
```

kernel/fs/file_reference.c

«

Si veda la sezione i159.3.13.

```
560001  #include <kernel/proc.h>
560002  #include <errno.h>
560003  #include <fcntl.h>
560004  //-----------------------------------------------------------------------
560005  file_t *
560006  file_reference (int fno)
560007  {
560008      //
560009      // Check type of request.
560010      //
560011      if (fno < 0)
560012        {
560013          //
560014          // Find a free slot.
560015          //
560016          for (fno = 0; fno < FILE_MAX_SLOTS; fno++)
560017            {
560018              if (file_table[fno].references <= 0)
560019                {
560020                  return (&file_table[fno]);
560021                }
560022            }
560023          return (NULL);
560024        }
560025      else if (fno > FILE_MAX_SLOTS)
560026        {
560027          return (NULL);
560028        }
560029      else
560030        {
560031          return (&file_table[fno]);
560032        }
560033  }
```


kernel/fs/file_stdio_dev_make.c

«

Si veda la sezione i159.3.14.

```
570001  #include <kernel/proc.h>
570002  #include <errno.h>
570003  #include <fcntl.h>
570004  //-----------------------------------------------------------------------
570005  file_t *
570006  file_stdio_dev_make (dev_t device, mode_t mode, int oflags)
570007  {
570008      inode_t *inode;
570009      file_t  *file;
570010      //
570011      // Try to allocate a device inode.
570012      //
570013      inode = inode_stdio_dev_make (device, mode);
570014      if (inode == NULL)
570015        {
570016          //
570017          // Variable 'errno' is already set by 'inode_stdio_dev_make()'.
570018          //
570019          errset (errno);
570020          return (NULL);
570021        }
570022      //
570023      // Inode allocated: need to allocate the system file item.
570024      //
570025      file = file_reference (-1);
570026      if (file == NULL)
570027        {
570028          //
570029          // Remove the inode and return an error.
570030          //
570031          inode_put (inode);
570032          errset (ENFILE);        // Too many files open in system.
570033          return (NULL);
570034        }
570035      //
570036      // Fill with data the system file item.
570037      //
570038      file->references = 1;
570039      file->oflags     = (oflags & (O_RDONLY | O_WRONLY));
570040      file->inode      = inode;
570041      //
570042      // Return system file pointer.
570043      //
570044      return (file);
570045  }
```

kernel/fs/file_table.c

«

Si veda la sezione i159.3.13.

```
580001  #include <kernel/fs.h>
580002  //-----------------------------------------------------------------------
580003  file_t file_table[FILE_MAX_SLOTS];
580004  //-----------------------------------------------------------------------
```

kernel/fs/inode_alloc.c

«

Si veda la sezione i159.3.15.

```
590001  #include <kernel/fs.h>
590002  #include <errno.h>
590003  #include <kernel/k_libc.h>
590004  //-----------------------------------------------------------------------
590005  inode_t *
590006  inode_alloc (dev_t device, mode_t mode, uid_t uid)
590007  {
590008      sb_t        *sb;
590009      inode_t     *inode;
590010      int          m;                     // Index inside the inode map.
590011      int          map_element;
590012      int          map_bit;
590013      int          map_mask;
590014      ino_t        ino;
590015      //
590016      // Check for arguments.
590017      //
590018      if (mode == 0)
590019        {
590020          errset (EINVAL);        // Invalid argument.
590021          return (NULL);
590022        }
590023      //
590024      // Get the super block from the known device.
590025      //
590026      sb = sb_reference (device);
590027      if (sb == NULL)
590028        {
590029          errset (ENODEV);        // No such device.
590030          return (NULL);
590031        }
590032      //
590033      // Find a free inode.
590034      //
590035      while (1)
590036        {
590037          //
```

```
590038          // Scan the inode bit map, to find a free inode
590039          // for new allocation.
590040          //
590041          for (m = 0; m < (SB_MAP_INODE_SIZE * 16); m++)
590042            {
590043              map_element = m / 16;
590044              map_bit     = m % 16;
590045              map_mask    = 1 << map_bit;
590046              if (!(sb->map_inode[map_element] & map_mask))
590047                {
590048                  //
590049                  // Found a free element: change the map to
590050                  // allocate the inode.
590051                  //
590052                  sb->map_inode[map_element] |= map_mask;
590053                  sb->changed = 1;
590054                  ino = m;  // Found a free inode:
590055                  break;          // exit the scan loop.
590056                }
590057            }
590058          //
590059          // Check if the scan was successful.
590060          //
590061          if (ino == 0)
590062            {
590063              errset (ENOSPC);    // No space left on device.
590064              return (NULL);
590065            }
590066          //
590067          // The inode was allocated inside the map in memory.
590068          //
590069          inode = inode_get (device, ino);
590070          if (inode == NULL)
590071            {
590072              errset (ENFILE);    // Too many files open in system.
590073              return (NULL);
590074            }
590075          //
590076          // Verify if the inode is really free: if it isn't, must save
590077          // it to disk.
590078          //
590079          if (inode->size > 0 || inode->links > 0)
590080            {
590081              //
590082              // Strange: should not have a size! Check if there are even
590083              // links. Please note that 255 links (that is -1) is to be
590084              // considered a free inode, marked in a special way for some
590085              // unknown reason. Currently, 'LINK_MAX' is equal to 254,
590086              // for that reason.
590087              //
590088              if (inode->links > 0 && inode->links < LINK_MAX)
590089                {
590090                  //
590091                  // Tell something.
590092                  //
590093                  k_printf ("kernel alert: device %04x: "
590094                            "found \"free\" inode %i "
590095                            "that still has size %i "
590096                            "and %i links!\n",
590097                            device, ino, inode->size, inode->links);
590098                  //
590099                  // The inode must be set again to free, inside
590100                  // the bit map.
590101                  //
590102                  map_element = ino / 16;
590103                  map_bit     = ino % 16;
590104                  map_mask    = 1 << map_bit;
590105                  sb->map_inode[map_element] &= ~map_mask;
590106                  sb->changed = 1;
590107                  //
590108                  // Try to fix: reset all to zero.
590109                  //
590110                  inode->mode      = 0;
590111                  inode->uid       = 0;
590112                  inode->gid       = 0;
590113                  inode->time      = 0;
590114                  inode->links     = 0;
590115                  inode->size      = 0;
590116                  inode->direct[0] = 0;
590117                  inode->direct[1] = 0;
590118                  inode->direct[2] = 0;
590119                  inode->direct[3] = 0;
590120                  inode->direct[4] = 0;
590121                  inode->direct[5] = 0;
590122                  inode->direct[6] = 0;
590123                  inode->indirect1 = 0;
590124                  inode->indirect2 = 0;
590125                  inode->changed   = 1;
590126                  //
590127                  // Save fixed inode to disk.
590128                  //
590129                  inode_put (inode);
590130                  continue;
590131                }
590132              else
590133                {
590134                  //
590135                  // Truncate the inode, save and break.
590136                  //
590137                  inode_truncate (inode);
590138                  inode_save (inode);
```



```
590139                  break;
590140                }
590141            }
590142          else
590143            {
590144              //
590145              // Considering free the inode found.
590146              //
590147              break;
590148            }
590149        }
590150      //
590151      // Put data inside the inode.
590152      //
590153      inode->mode    = mode;
590154      inode->uid     = uid;
590155      inode->gid     = 0;
590156      inode->size    = 0;
590157      inode->time    = k_time (NULL);
590158      inode->links   = 0;
590159      inode->changed = 1;
590160      //
590161      // Save the inode.
590162      //
590163      inode_save (inode);
590164      //
590165      // Return the inode pointer.
590166      //
590167      return (inode);
590168  }
```

kernel/fs/inode_check.c

Si veda la sezione i159.3.16.

```
600001  #include <kernel/fs.h>
600002  #include <errno.h>
600003  #include <kernel/k_libc.h>
600004  //---------------------------------------------------------------------
600005  int
600006  inode_check (inode_t *inode, mode_t type, int perm, uid_t uid)
600007  {
600008      //
600009      // Ensure that the variable 'type' has only the requested file type.
600010      //
600011      type = (type & S_IFMT);
600012      //
600013      // Check inode argument.
600014      //
600015      if (inode == NULL)
600016        {
600017          errset (EINVAL);        // Invalid argument.
600018          return (-1);
600019        }
600020      //
600021      // The inode is not NULL: verify that the inode is of a type
600022      // allowed (the parameter 'type' can hold more than one
600023      // possibility).
600024      //
600025      if (!(inode->mode & type))
600026        {
600027          errset (E_FILE_TYPE);   // The file type is not
600028          return (-1);            // the expected one.
600029        }
600030      //
600031      // The file type is correct.
600032      //
600033      if (inode->uid != 0 && uid == 0)
600034        {
600035          return (0);             // The root user has all permissions.
600036        }
600037      //
600038      // The user is not root or the inode is owned by root.
600039      //
600040      if (inode->uid == uid)
600041        {
600042          //
600043          // The user own the inode and must check user permissions.
600044          //
600045          perm = (perm << 6);
600046          if ((inode->mode & perm) ^ perm)
600047            {
600048              errset (EACCES);            // Permission denied.
600049              return (-1);
600050            }
600051          else
600052            {
600053              return (0);
600054            }
600055        }
600056      //
600057      // The user does not own the inode: the other permissions are
600058      // checked.
600059      //
600060      if ((inode->mode & perm) ^ perm)
600061        {
600062          errset (EACCES);                // Permission denied.
600063          return (-1);
600064        }
600065      else
600066        {
```

```
600067          return (0);
600068        }
600069  }
```

kernel/fs/inode_dir_empty.c

«

Si veda la sezione i159.3.17.

```
610001  #include <kernel/fs.h>
610002  #include <errno.h>
610003  #include <kernel/k_libc.h>
610004  //---------------------------------------------------------------------
610005  int
610006  inode_dir_empty (inode_t *inode)
610007  {
610008      off_t        start;
610009      char         buffer[SB_MAX_ZONE_SIZE];
610010      directory_t *dir;
610011      ssize_t      size_read;
610012      int          d;                     // Directory buffer index.
610013      //
610014      // Check argument: must be a directory.
610015      //
610016      if (inode == NULL || !S_ISDIR (inode->mode))
610017        {
610018          errset (EINVAL);                // Invalid argument.
610019          return (0);                     // false
610020        }
610021      //
610022      // Read the directory content: if an item is present (except '.' and
610023      // '..'), the directory is not empty.
610024      //
610025      for (start  = 0;
610026           start <  inode->size;
610027           start += inode->sb->blksize)
610028        {
610029          size_read = inode_file_read (inode, start, buffer,
610030                                       inode->sb->blksize,
610031                                       NULL);
610032          if (size_read < sizeof (directory_t))
610033            {
610034              break;
610035            }
610036          //
610037          // Scan the directory portion just read.
610038          //
610039          dir = (directory_t *) buffer;
610040          //
610041          for (d = 0; d < size_read; d += (sizeof (directory_t)), dir++)
610042            {
610043              if (dir->ino != 0                                   &&
610044                  strncmp (dir->name, ".", NAME_MAX) != 0    &&
610045                  strncmp (dir->name, "..", NAME_MAX) != 0)
610046                {
610047                  //
610048                  // There is an item and the directory is not empty.
610049                  //
610050                  return (0);     // false
610051                }
610052            }
610053        }
610054      //
610055      // Nothing was found; good!
610056      //
610057      return (1);         // true
610058  }
```

kernel/fs/inode_file_read.c

«

Si veda la sezione i159.3.18.

```
620001  #include <kernel/fs.h>
620002  #include <errno.h>
620003  #include <kernel/k_libc.h>
620004  //---------------------------------------------------------------------
620005  ssize_t
620006  inode_file_read (inode_t *inode, off_t offset,
620007                   void *buffer, size_t count, int *eof)
620008  {
620009      unsigned char *destination = (unsigned char *) buffer;
620010      unsigned char  zone_buffer[SB_MAX_ZONE_SIZE];
620011      blkcnt_t       blkcnt_read;
620012      off_t          off_fzone;   // File zone offset.
620013      off_t          off_buffer;  // Destination buffer offset.
620014      ssize_t        size_read;   // Byte transfer counter.
620015      zno_t          fzone;
620016      off_t          off_end;
620017      //
620018      // The inode pointer must be valid, and
620019      // the start byte must be positive.
620020      //
620021      if (inode == NULL || offset < 0)
620022        {
620023          errset (EINVAL);        // Invalid argument.
620024          return ((ssize_t) -1);
620025        }
620026      //
620027      // Check if the start address is inside the file size. This is not
620028      // an error, but zero bytes are read and '*eof' is set. Otherwise,
620029      // '*eof' is reset.
620030      //
620031      if (offset >= inode->size)
620032        {
620033          (eof != NULL)? *eof = 1: 0;
620034          return (0);
620035        }
620036      else
620037        {
620038          (eof != NULL)? *eof = 0: 0;
620039        }
620040      //
620041      // Adjust, if necessary, the size of read, because it cannot be
620042      // larger than the actual file size. The variable 'off_end' is
620043      // used to calculate the position *after* the requested read.
620044      // Remember that the first file position is byte zero; so,
620045      // the byte index inside the file goes from zero to inode->size -1.
620046      //
620047      off_end  = offset;
620048      off_end += count;
620049      if (off_end > inode->size)
620050        {
620051          count = (inode->size - offset);
620052        }
620053      //
620054      // Read the first file-zone inside the zone buffer.
620055      //
620056      fzone       = offset / inode->sb->blksize;
620057      off_fzone   = offset % inode->sb->blksize;
620058      blkcnt_read = inode_fzones_read (inode, fzone, zone_buffer,
620059                                       (blkcnt_t) 1);
620060      if (blkcnt_read <= 0)
620061        {
620062          //
620063          // Sorry!
620064          //
620065          return (0);                 // Zero bytes read!
620066        }
620067      //
620068      // The first file-zone was read: copy it inside the destination
620069      // buffer and continue reading the other zones needed. Variables
620070      // 'off_buffer' (destination buffer index) and 'size_read' (copy
620071      // byte counter) must be reset here. Variable 'off_fzone' is already
620072      // set with the initial offset inside 'zone_buffer'.
620073      //
620074      off_buffer = 0;
620075      size_read  = 0;
620076      //
620077      while (count)
620078        {
620079          //
620080          // Copy the zone buffer into the destination. Variables
620081          // 'off_fzone', 'off_buffer' and 'size_read' must not be
620082          // initialized inside the loop.
620083          //
620084          for (; off_fzone < inode->sb->blksize && count > 0;
620085               off_fzone++, off_buffer++, size_read++,
620086               count--, offset++)
620087            {
620088              destination[off_buffer] = zone_buffer[off_fzone];
620089            }
620090          //
620091          // If not all the bytes are copied, read the next file-zone.
620092          //
620093          if (count)
620094            {
620095              //
620096              // Read another file-zone inside the zone buffer.
620097              // Again, the function 'inode_fzones_read()' might
620098              // return a null pointer, but the variable 'errno' tells if
620099              // it is really an error. For this reason, the variable
620100              // 'errno' must be reset before the read, and checked after
620101              // it.
620102              //
620103              fzone       = offset / inode->sb->blksize;
620104              off_fzone   = offset % inode->sb->blksize;
620105              blkcnt_read = inode_fzones_read (inode, fzone, zone_buffer,
620106                                               (blkcnt_t) 1);
620107              if (blkcnt_read <= 0)
620108                {
620109                  //
620110                  // Sorry: only 'size_read' bytes read!
620111                  //
620112                  return (size_read);
620113                }
620114            }
620115        }
620116      //
620117      // The requested size was read completely.
620118      //
620119      return (size_read);
620120  }
```

kernel/fs/inode_file_write.c

«

Si veda la sezione i159.3.19.

```
630001  #include <kernel/fs.h>
630002  #include <errno.h>
630003  #include <kernel/k_libc.h>
630004  //---------------------------------------------------------------------
630005  ssize_t
```

```
630006 inode_file_write (inode_t *inode, off_t offset, void *buffer,
630007                   size_t count)
630008 {
630009     unsigned char *buffer_source = (unsigned char *) buffer;
630010     unsigned char  buffer_zone[SB_MAX_ZONE_SIZE];
630011     off_t          off_fzone;    // File zone offset.
630012     off_t          off_source;   // Source buffer offset.
630013     ssize_t        size_copied;  // Byte transfer counter.
630014     ssize_t        size_written; // Byte written counter.
630015     zno_t          fzone;
630016     zno_t          zone;
630017     blkcnt_t       blkcnt_read;
630018     int            status;
630019     //
630020     // The inode pointer must be valid, and
630021     // the start byte must be positive.
630022     //
630023     if (inode == NULL || offset < 0)
630024       {
630025         errset (EINVAL);        // Invalid argument.
630026         return ((ssize_t) -1);
630027       }
630028     //
630029     // Read a zone, modify it with the source buffer, then write it back
630030     // and continue reading and writing other zones if needed.
630031     //
630032     for (size_written = 0, off_source = 0, size_copied = 0;
630033          count > 0; size_written += size_copied)
630034       {
630035         //
630036         // Read the next file-zone inside the zone buffer: the function
630037         // 'inode_zone()' is used to create automatically the zone, if
630038         // it does not exist.
630039         //
630040         fzone     = offset / inode->sb->blksize;
630041         off_fzone = offset % inode->sb->blksize;
630042         zone      = inode_zone (inode, fzone, 1);
630043         if (zone == 0)
630044           {
630045             //
630046             // Return previously written bytes. The variable 'errno' is
630047             // already set by 'inode_zone()'.
630048             //
630049             return (size_written);
630050           }
630051         blkcnt_read = inode_fzones_read (inode, fzone, buffer_zone,
630052                                          (blkcnt_t) 1);
630053         if (blkcnt_read <= 0)
630054           {
630055             //
630056             // Even if the value is zero, there is a problem reading the
630057             // zone to be overwritten (because 'inode_zone()' should
630058             // have already created such zone). The variable 'errno' is
630059             // already set by 'inode_fzones_read()'.
630060             //
630061             return ((ssize_t) -1);
630062           }
630063         //
630064         // The zone was successfully loaded inside the buffer: overwrite
630065         // the zone buffer with the source buffer.
630066         //
630067         for (size_copied = 0;
630068              off_fzone < inode->sb->blksize && count > 0;
630069              off_fzone++, off_source++, size_copied++, count--,
630070              offset++)
630071           {
630072             buffer_zone[off_fzone] = buffer_source[off_source];
630073           }
630074         //
630075         // Save the zone.
630076         //
630077         status = zone_write (inode->sb, zone, buffer_zone);
630078         if (status != 0)
630079           {
630080             //
630081             // Cannot save the zone: return the size already written.
630082             // The variable 'errno' is already set by 'zone_write()'.
630083             //
630084             return (size_written);
630085           }
630086         //
630087         // Zone saved: update the file size if necessary (and the inode
630088         // too).
630089         //
630090         if (inode->size <= offset)
630091           {
630092             inode->size    = offset;
630093             inode->changed = 1;
630094             inode_save (inode);
630095           }
630096       }
630097     //
630098     // All done successfully: return the value.
630099     //
630100     return (size_written);
630101 }
```


kernel/fs/inode_free.c

«

Si veda la sezione i159.3.20.

```
640001 #include <kernel/fs.h>
640002 #include <errno.h>
640003 #include <kernel/k_libc.h>
640004 //-----------------------------------------------------------------------
640005 int
640006 inode_free (inode_t *inode)
640007 {
640008     int      map_element;
640009     int      map_bit;
640010     int      map_mask;
640011     //
640012     if (inode == NULL)
640013       {
640014         errset (EINVAL);        // Invalid argument.
640015         return (-1);
640016       }
640017     //
640018     map_element = inode->ino / 16;
640019     map_bit     = inode->ino % 16;
640020     map_mask    = 1 << map_bit;
640021     //
640022     if (inode->sb->map_inode[map_element] & map_mask)
640023       {
640024         inode->sb->map_inode[map_element] -= map_mask;
640025         inode->sb->changed = 1;
640026       }
640027     //
640028     inode->mode       = 0;
640029     inode->uid        = 0;
640030     inode->gid        = 0;
640031     inode->size       = 0;
640032     inode->time       = 0;
640033     inode->links      = 0;
640034     inode->changed    = 1;
640035     inode->references = 0;
640036     //
640037     return (inode_save (inode));
640038 }
```

kernel/fs/inode_fzones_read.c

«

Si veda la sezione i159.3.21.

```
650001 #include <kernel/fs.h>
650002 #include <errno.h>
650003 #include <kernel/k_libc.h>
650004 //-----------------------------------------------------------------------
650005 blkcnt_t
650006 inode_fzones_read (inode_t *inode, zno_t zone_start,
650007                    void *buffer, blkcnt_t blkcnt)
650008 {
650009     unsigned char *destination = (unsigned char *) buffer;
650010     int            status;        // 'zone_read()' return value.
650011     blkcnt_t       blkcnt_read;   // Zone counter/index.
650012     zno_t          zone;
650013     zno_t          fzone;
650014     //
650015     // Read the zones into the destination buffer.
650016     //
650017     for (blkcnt_read = 0, fzone = zone_start;
650018          blkcnt_read < blkcnt;
650019          blkcnt_read++, fzone++)
650020       {
650021         //
650022         // Calculate the zone number, from the file-zone, reading the
650023         // inode. If a zone is not really allocated, the result is zero
650024         // and is valid.
650025         //
650026         zone = inode_zone (inode, fzone, 0);
650027         if (zone == ((zno_t) -1))
650028           {
650029             //
650030             // This is an error. Return the read zones quantity.
650031             //
650032             return (blkcnt_read);
650033           }
650034         //
650035         // Update the destination buffer pointer.
650036         //
650037         destination += (blkcnt_read * inode->sb->blksize);
650038         //
650039         // Read the zone inside the destination buffer, but if the zone
650040         // is zero, a zeroed zone must be filled.
650041         //
650042         if (zone == 0)
650043           {
650044             memset (destination, 0, (size_t) inode->sb->blksize);
650045           }
650046         else
650047           {
650048             status = zone_read (inode->sb, zone, destination);
650049             if (status != 0)
650050               {
650051                 //
650052                 // Could not read the requested zone: return the zones
650053                 // read correctly.
650054                 //
```

```
650055                  errset (EIO);               // I/O error.
650056                  return (blkcnt_read);
650057                }
650058            }
650059        }
650060      //
650061      // All zones read correctly inside the buffer.
650062      //
650063      return (blkcnt_read);
650064  }
```

kernel/fs/inode_fzones_write.c

«

Si veda la sezione i159.3.21.

```
660001  #include <kernel/fs.h>
660002  #include <errno.h>
660003  #include <kernel/k_libc.h>
660004  //-----------------------------------------------------------------------
660005  blkcnt_t
660006  inode_fzones_write (inode_t *inode, zno_t zone_start, void *buffer,
660007                      blkcnt_t blkcnt)
660008  {
660009      unsigned char *source = (unsigned char *) buffer;
660010      int            status;              // 'zone_read()' return value.
660011      blkcnt_t       blkcnt_written;      // Written zones counter.
660012      zno_t          zone;
660013      zno_t          fzone;
660014      //
660015      // Write the zones into the destination buffer.
660016      //
660017      for (blkcnt_written = 0, fzone = zone_start;
660018           blkcnt_written < blkcnt;
660019           blkcnt_written++, fzone++)
660020        {
660021          //
660022          // Find real zone from file-zone.
660023          //
660024          zone = inode_zone (inode, fzone, 1);
660025          if (zone == 0 || zone == ((zno_t) -1))
660026            {
660027              //
660028              // Function 'inode_zone()' should allocate automatically
660029              // a missing zone and should return a valid zone or
660030              // (zno_t) -1. Anyway, even if a zero zone is returned,
660031              // it is an error. Return the 'blkcnt_written' value.
660032              //
660033              return (blkcnt_written);
660034            }
660035          //
660036          // Update the source buffer pointer for the next zone write.
660037          //
660038          source += (blkcnt_written * inode->sb->blksize);
660039          //
660040          // Write the zone from the buffer content.
660041          //
660042          status  = zone_write (inode->sb, zone, source);
660043          if (status != 0)
660044            {
660045              //
660046              // Cannot write the zone. Return 'size_written_zone' value.
660047              //
660048              return (blkcnt_written);
660049            }
660050        }
660051      //
660052      // All zones read correctly inside the buffer.
660053      //
660054      return (blkcnt_written);
660055  }
```

kernel/fs/inode_get.c

«

Si veda la sezione i159.3.23.

```
670001  #include <kernel/fs.h>
670002  #include <errno.h>
670003  #include <kernel/k_libc.h>
670004  #include <kernel/devices.h>
670005  //-----------------------------------------------------------------------
670006  inode_t *
670007  inode_get (dev_t device, ino_t ino)
670008  {
670009      sb_t             *sb;
670010      inode_t          *inode;
670011      unsigned long int start;
670012      size_t            size;
670013      ssize_t           n;
670014      int               status;
670015      //
670016      // Verify if the root file system inode was requested.
670017      //
670018      if (device == 0 && ino == 1)
670019        {
670020          //
670021          // Get root file system inode.
670022          //
670023          inode = inode_reference (device, ino);
670024          if (inode == NULL)
670025            {
```



```
670026              //
670027              // The file system root directory inode is not yet loaded:
670028              // get the first super block.
670029              //
670030              sb = sb_reference ((dev_t) 0);
670031              if (sb == NULL || sb->device == 0)
670032                {
670033                  //
670034                  // This error should never happen.
670035                  //
670036                  errset (EUNKNOWN);      // Unknown error.
670037                  return (NULL);
670038                }
670039              //
670040              // Load the file system root directory inode (recursive
670041              // call).
670042              //
670043              inode = inode_get (sb->device, (ino_t) 1);
670044              if (inode == NULL)
670045                {
670046                  //
670047                  // This error should never happen.
670048                  //
670049                  return (NULL);
670050                }
670051              //
670052              // Return the directory inode.
670053              //
670054              return (inode);
670055            }
670056          else
670057            {
670058              //
670059              // The file system root directory inode is already
670060              // available.
670061              //
670062              if (inode->references >= INODE_MAX_REFERENCES)
670063                {
670064                  errset (ENFILE); // Too many files open in system.
670065                  return (NULL);
670066                }
670067              else
670068                {
670069                  inode->references++;
670070                  return (inode);
670071                }
670072            }
670073        }
670074      //
670075      // A common device-inode pair was requested: try to find an already
670076      // cached inode.
670077      //
670078      inode = inode_reference (device, ino);
670079      if (inode != NULL)
670080        {
670081          if (inode->references >= INODE_MAX_REFERENCES)
670082            {
670083              errset (ENFILE);    // Too many files open in system.
670084              return (NULL);
670085            }
670086          else
670087            {
670088              inode->references++;
670089              return (inode);
670090            }
670091        }
670092      //
670093      // The inode is not yet available: get super block.
670094      //
670095      sb = sb_reference (device);
670096      if (sb == NULL)
670097        {
670098          errset (ENODEV);        // No such device.
670099          return (NULL);
670100        }
670101      //
670102      // The super block is available, but the inode is not yet cached.
670103      // Verify if the inode map reports it as allocated.
670104      //
670105      status = sb_inode_status (sb, ino);
670106      if (!status)
670107        {
670108          //
670109          // The inode is not allocated and cannot be loaded.
670110          //
670111          errset (ENOENT);        // No such file or directory.
670112          return (NULL);
670113        }
670114      //
670115      // The inode was not already cached, but is considered as allocated
670116      // inside the inode map. Find a free slot to load the inode inside
670117      // the inode table (in memory).
670118      //
670119      inode = inode_reference ((dev_t) -1, (ino_t) -1);
670120      if (inode == NULL)
670121        {
670122          errset (ENFILE);        // Too many files open in system.
670123          return (NULL);
670124        }
670125      //
670126      // A free inode slot was found. The inode must be loaded.
```

```
670127      // Calculate the memory inode size, to be saved inside the file
670128      // system: the administrative inode data, as it is saved inside
670129      // the file system. The 'inode_t' type is bigger than the real
670130      // inode administrative size, because it contains more data, that is
670131      // not saved on disk.
670132      //
670133      size   = offsetof (inode_t, sb);
670134      //
670135      // Calculating start position for read.
670136      //
670137      // [1] Boot block.
670138      // [2] Super block.
670139      // [3] Inode bit map.
670140      // [4] Zone bit map.
670141      // [5] Previous inodes: consider that the inode zero is
670142      //     present in the inode map, but not in the inode
670143      //     table.
670144      //
670145      start  = 1024;                          // [1]
670146      start += 1024;                          // [2]
670147      start += (sb->map_inode_blocks * 1024);  // [3]
670148      start += (sb->map_zone_blocks * 1024);   // [4]
670149      start += ((ino -1) * size);              // [5]
670150      //
670151      // Read inode from disk.
670152      //
670153      n = dev_io ((pid_t) -1, device, DEV_READ, start, inode, size, NULL);
670154      if (n != size)
670155        {
670156          errset (EIO);                   // I/O error.
670157          return (NULL);
670158        }
670159      //
670160      // The inode was read: add some data to the working copy in memory.
670161      //
670162      inode->sb            = sb;
670163      inode->sb_attached   = NULL;
670164      inode->ino           = ino;
670165      inode->references    = 1;
670166      inode->changed       = 0;
670167      //
670168      inode->blkcnt        = inode->size;
670169      inode->blkcnt       /= sb->blksize;
670170      if (inode->size % sb->blksize)
670171        {
670172          inode->blkcnt++;
670173        }
670174      //
670175      // Return the inode pointer.
670176      //
670177      return (inode);
670178  }
```

kernel/fs/inode_put.c

«

Si veda la sezione i159.3.24.

```
680001  #include <kernel/fs.h>
680002  #include <errno.h>
680003  #include <kernel/k_libc.h>
680004  //------------------------------------------------------------------------
680005  int
680006  inode_put (inode_t *inode)
680007  {
680008      int        status;
680009      //
680010      // Check for valid argument.
680011      //
680012      if (inode == NULL)
680013        {
680014          errset (EINVAL);                // Invalid argument.
680015          return (-1);
680016        }
680017      //
680018      // Check for valid references.
680019      //
680020      if (inode->references <= 0)
680021        {
680022          errset (EUNKNOWN);              // Cannot put an inode with
680023          return (-1);                    // zero or negative references.
680024        }
680025      //
680026      // Debug.
680027      //
680028      if (inode->sb->device == 0 && inode->ino != 0)
680029        {
680030          k_printf ("kernel alert: trying to close inode with device "
680031                    "zero, but a number different than zero!\n");
680032          errset (EUNKNOWN);              // Cannot put an inode with
680033          return (-1);                    // zero or negative references.
680034        }
680035      //
680036      // There is at least one reference: now the references value is
680037      // reduced.
680038      //
680039      inode->references--;
680040      inode->changed = 1;
680041      //
680042      // If 'inode->ino' is zero, it means that the inode was created in
680043      // memory, but there is no file system for it. For example, it might
680044      // be a standard I/O inode create automatically for a process.
```



```
680045      // Inodes with number zero cannot be removed from a file system.
680046      //
680047      if (inode->ino == 0)
680048        {
680049          //
680050          // Nothing to do: just return.
680051          //
680052          return (0);
680053        }
680054      //
680055      //  References counter might be zero.
680056      //
680057      if (inode->references == 0)
680058        {
680059          //
680060          // Check if the inode is to be deleted (until there are
680061          // run time references, the inode cannot be removed).
680062          //
680063          if (inode->links == 0
680064           || (S_ISDIR (inode->mode) && inode->links == 1))
680065            {
680066              //
680067              // The inode has no more run time references and file system
680068              // links are also zero (or one for a directory): remove it!
680069              //
680070              status = inode_truncate (inode);
680071              if (status != 0)
680072                {
680073                  k_perror (NULL);
680074                }
680075              //
680076              inode_free (inode);
680077              return (0);
680078            }
680079        }
680080      //
680081      // Save inode to disk and return.
680082      //
680083      return (inode_save (inode));
680084  }
```

kernel/fs/inode_reference.c

«

Si veda la sezione i159.3.25.

```
690001  #include <kernel/fs.h>
690002  #include <errno.h>
690003  #include <kernel/k_libc.h>
690004  //------------------------------------------------------------------------
690005  inode_t *
690006  inode_reference (dev_t device, ino_t ino)
690007  {
690008      int   s;                            // Slot index.
690009      sb_t *sb_table = sb_reference (0);
690010      //
690011      // If device is zero, and inode is zero, a reference to the whole
690012      // table is returned.
690013      //
690014      if (device == 0 && ino == 0)
690015        {
690016          return (inode_table);
690017        }
690018      //
690019      // If device is ((dev_t) -1) and the inode is ((ino_t) -1), a
690020      // reference to a free inode slot is returned.
690021      //
690022      if (device == (dev_t) -1 && ino == ((ino_t) -1))
690023        {
690024          for (s = 0; s < INODE_MAX_SLOTS; s++)
690025            {
690026              if (inode_table[s].references == 0)
690027                {
690028                  return (&inode_table[s]);
690029                }
690030            }
690031          return (NULL);
690032        }
690033      //
690034      // If device is zero and the inode is 1, a reference to the root
690035      // directory inode is returned.
690036      //
690037      if (device == 0 && ino == 1)
690038        {
690039          //
690040          // The super block table is to be scanned.
690041          //
690042          for (device = 0, s = 0; s < SB_MAX_SLOTS; s++)
690043            {
690044              if (sb_table[s].device != 0                &&
690045                  sb_table[s].inode_mounted_on == NULL)
690046                {
690047                  device = sb_table[s].device;
690048                  break;
690049                }
690050            }
690051          if (device == 0)
690052            {
690053              errset (E_CANNOT_FIND_ROOT_DEVICE);
690054              return (NULL);
690055            }
690056          //
```

```
690057          // Scan the inode table to find inode 1 and the same device.
690058          //
690059          for (s = 0; s < INODE_MAX_SLOTS; s++)
690060            {
690061              if (inode_table[s].sb->device == device     &&
690062                  inode_table[s].ino == 1)
690063                {
690064                  return (&inode_table[s]);
690065                }
690066            }
690067          //
690068          // Cannot find a root file system inode.
690069          //
690070          errset (E_CANNOT_FIND_ROOT_INODE);
690071          return (NULL);
690072        }
690073      //
690074      // A device and an inode number were selected: find the inode
690075      // associated to it.
690076      //
690077      for (s = 0; s < INODE_MAX_SLOTS; s++)
690078        {
690079          if (inode_table[s].sb->device == device &&
690080              inode_table[s].ino == ino)
690081            {
690082              return (&inode_table[s]);
690083            }
690084        }
690085      //
690086      // The inode was not found.
690087      //
690088      return (NULL);
690089  }
```

kernel/fs/inode_save.c

«

Si veda la sezione i159.3.26.

```
700001  #include <kernel/fs.h>
700002  #include <errno.h>
700003  #include <kernel/k_libc.h>
700004  #include <kernel/devices.h>
700005  //-------------------------------------------------------------------------
700006  int
700007  inode_save (inode_t *inode)
700008  {
700009      size_t            size;
700010      unsigned long int start;
700011      ssize_t           n;
700012      //
700013      // Check for valid argument.
700014      //
700015      if (inode == NULL)
700016        {
700017          errset (EINVAL);               // Invalid argument.
700018          return (-1);
700019        }
700020      //
700021      // If the inode number is zero, no file system is involved!
700022      //
700023      if (inode->ino == 0)
700024        {
700025          return (0);
700026        }
700027      //
700028      // Save the super block to disk.
700029      //
700030      sb_save (inode->sb);
700031      //
700032      // Save the inode to disk.
700033      //
700034      if (inode->changed)
700035        {
700036          size   = offsetof (inode_t, sb);
700037          //
700038          // Calculating start position for write.
700039          //
700040          // [1] Boot block.
700041          // [2] Super block.
700042          // [3] Inode bit map.
700043          // [4] Zone bit map.
700044          // [5] Previous inodes: consider that the inode zero is
700045          //     present in the inode map, but not in the inode
700046          //     table.
700047          //
700048          start  = 1024;                                  // [1]
700049          start += 1024;                                  // [2]
700050          start += (inode->sb->map_inode_blocks * 1024);  // [3]
700051          start += (inode->sb->map_zone_blocks * 1024);   // [4]
700052          start += ((inode->ino -1) * size);              // [5]
700053          //
700054          // Write the inode.
700055          //
700056          n = dev_io ((pid_t) -1, inode->sb->device, DEV_WRITE, start,
700057                      inode, size, NULL);
700058          //
700059          inode->changed = 0;
700060        }
700061      return (0);
700062  }
```


kernel/fs/inode_stdio_dev_make.c

«

Si veda la sezione i159.3.27.

```
710001  #include <kernel/fs.h>
710002  #include <errno.h>
710003  #include <kernel/k_libc.h>
710004  //-------------------------------------------------------------------------
710005  inode_t *
710006  inode_stdio_dev_make (dev_t device, mode_t mode)
710007  {
710008      inode_t     *inode;
710009      //
710010      // Check for arguments.
710011      //
710012      if (mode == 0 || device == 0)
710013        {
710014          errset (EINVAL);        // Invalid argument.
710015          return (NULL);
710016        }
710017      //
710018      // Find a free inode.
710019      //
710020      inode = inode_reference ((dev_t) -1, (ino_t) -1);
710021      if (inode == NULL)
710022        {
710023          //
710024          // No free slot available.
710025          //
710026          errset (ENFILE);        // Too many files open in system.
710027          return (NULL);
710028        }
710029      //
710030      // Put data inside the inode. Please note that 'inode->ino' must be
710031      // zero, because it is necessary to recognize it as an internal
710032      // inode with no file system. Otherwise, with a value different than
710033      // zero, 'inode_put()' will try to remove it. [*]
710034      //
710035      inode->mode          = mode;
710036      inode->uid           = 0;
710037      inode->gid           = 0;
710038      inode->size          = 0;
710039      inode->time          = k_time (NULL);
710040      inode->links         = 0;
710041      inode->direct[0]     = device;
710042      inode->direct[1]     = 0;
710043      inode->direct[2]     = 0;
710044      inode->direct[3]     = 0;
710045      inode->direct[4]     = 0;
710046      inode->direct[5]     = 0;
710047      inode->direct[6]     = 0;
710048      inode->indirect1     = 0;
710049      inode->indirect2     = 0;
710050      inode->sb_attached   = NULL;
710051      inode->sb            = 0;
710052      inode->ino           = 0;   // Must be zero. [*]
710053      inode->blkcnt        = 0;
710054      inode->references    = 1;
710055      inode->changed       = 0;
710056      //
710057      // Add all access permissions.
710058      //
710059      inode->mode       |= (S_IRWXU|S_IRWXG|S_IRWXO);
710060      //
710061      // Return the inode pointer.
710062      //
710063      return (inode);
710064  }
```

kernel/fs/inode_table.c

«

Si veda la sezione i159.3.25.

```
720001  #include <kernel/fs.h>
720002  //-------------------------------------------------------------------------
720003  inode_t inode_table[INODE_MAX_SLOTS];
```

kernel/fs/inode_truncate.c

«

Si veda la sezione i159.3.28.

```
730001  #include <kernel/fs.h>
730002  #include <errno.h>
730003  #include <kernel/k_libc.h>
730004  //-------------------------------------------------------------------------
730005  int
730006  inode_truncate (inode_t *inode)
730007  {
730008    unsigned int indirect_zones;
730009    zno_t        zone_table1[INODE_MAX_INDIRECT_ZONES];
730010    zno_t        zone_table2[INODE_MAX_INDIRECT_ZONES];
730011    unsigned int i;             // Direct index.
730012    unsigned int i0;            // Single indirect index.
730013    unsigned int i1;            // Double indirect first index.
730014    unsigned int i2;            // Double indirect second index.
730015    int          status;        // 'zone_read()' return value.
730016    //
730017    // Calculate how many indirect zone numbers are stored inside
730018    // a zone: it depends on the zone size.
730019    //
```

```
730020    indirect_zones  = inode->sb->blksize / 2;
730021    //
730022    // Scan and release direct zones. Errors are ignored.
730023    //
730024    for (i = 0; i < 7; i++)
730025      {
730026        zone_free (inode->sb, inode->direct[i]);
730027        inode->direct[i] = 0;
730028      }
730029    //
730030    // Scan single indirect zones, if present.
730031    //
730032    if (inode->blkcnt > 7 && inode->indirect1 != 0)
730033      {
730034        //
730035        // There is a single indirect table to load. Errors are
730036        // almost ignored.
730037        //
730038        status = zone_read (inode->sb, inode->indirect1, zone_table1);
730039        if (status == 0)
730040          {
730041            //
730042            // Scan the table and remove zones.
730043            //
730044            for (i0 = 0; i0 < indirect_zones; i0++)
730045              {
730046                zone_free (inode->sb, zone_table1[i0]);
730047              }
730048          }
730049        //
730050        // Remove indirect table too.
730051        //
730052        zone_free (inode->sb, inode->indirect1);
730053        //
730054        // Clear single indirect reference inside the inode.
730055        //
730056        inode->indirect1 = 0;
730057      }
730058    //
730059    // Scan double indirect zones, if present.
730060    //
730061    if (   inode->blkcnt > (7+indirect_zones)
730062        && inode->indirect2 != 0)
730063      {
730064        //
730065        // There is a double indirect table to load. Errors are
730066        // almost ignored.
730067        //
730068        status = zone_read (inode->sb, inode->indirect2, zone_table1);
730069        if (status == 0)
730070          {
730071            //
730072            // Scan the table and get second level indirection.
730073            //
730074            for (i1 = 0; i1 < indirect_zones; i1++)
730075              {
730076                if ((inode->blkcnt > (7+indirect_zones+indirect_zones*i1))
730077                    && zone_table1[i1] != 0)
730078                  {
730079                    //
730080                    // There is a second level table to load.
730081                    //
730082                    status = zone_read (inode->sb, zone_table1[i1],
730083                                        zone_table2);
730084                    if (status == 0)
730085                      {
730086                        //
730087                        // Release zones.
730088                        //
730089                        for (i2 = 0;
730090                             i2 < indirect_zones &&
730091    (inode->blkcnt > (7+indirect_zones+indirect_zones*i1+i2));
730092                             i2++)
730093                          {
730094                            zone_free (inode->sb, zone_table2[i2]);
730095                          }
730096                        //
730097                        // Remove second level indirect table.
730098                        //
730099                        zone_free (inode->sb, zone_table1[i1]);
730100                      }
730101                  }
730102              }
730103            //
730104            // Remove first level indirect table.
730105            //
730106            zone_free (inode->sb, inode->indirect2);
730107          }
730108        //
730109        // Clear single indirect reference inside the inode.
730110        //
730111        inode->indirect2 = 0;
730112      }
730113    //
730114    // Update super block and inode data.
730115    //
730116    sb_save (inode->sb);
730117    inode->size = 0;
730118    inode->changed = 1;
730119    inode_save (inode);
730120    //
```



```
730121    // Successful return.
730122    //
730123    return (0);
730124  }
```

kernel/fs/inode_zone.c

«

Si veda la sezione i159.3.29.

```
740001  #include <kernel/fs.h>
740002  #include <errno.h>
740003  #include <kernel/k_libc.h>
740004  //---------------------------------------------------------------------
740005  zno_t
740006  inode_zone (inode_t *inode, zno_t fzone, int write)
740007  {
740008    unsigned int indirect_zones;
740009    unsigned int allocated_zone;
740010    zno_t        zone_table[INODE_MAX_INDIRECT_ZONES];
740011    char         buffer[SB_MAX_ZONE_SIZE];
740012    unsigned int i0;            // Single indirect index.
740013    unsigned int i1;            // Double indirect first index.
740014    unsigned int i2;            // Double indirect second index.
740015    int          status;
740016    zno_t        zone_second;   // Second level table zone.
740017    //
740018    // Calculate how many indirect zone numbers are stored inside
740019    // a zone: it depends on the zone size.
740020    //
740021    indirect_zones  = inode->sb->blksize / 2;
740022    //
740023    // Convert file-zone number into a zone number.
740024    //
740025    if (fzone < 7)
740026      {
740027        //
740028        // 0 <= fzone <= 6
740029        // The zone number is inside the direct zone references.
740030        // Verify to have such zone.
740031        //
740032        if (inode->direct[fzone] == 0)
740033          {
740034            //
740035            // There is not such zone, but we do not consider
740036            // it an error, because a file can be not contiguous.
740037            //
740038            if (!write)
740039              {
740040                return ((zno_t) 0);
740041              }
740042            //
740043            // Must be allocated.
740044            //
740045            allocated_zone = zone_alloc (inode->sb);
740046            if (allocated_zone == 0)
740047              {
740048                //
740049                // Cannot allocate the zone. The variable ‘errno’ is
740050                // set by ‘zone_alloc()’.
740051                //
740052                return ((zno_t) -1);
740053              }
740054            //
740055            // The zone is allocated: clear the zone and save.
740056            //
740057            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740058            status = zone_write (inode->sb, allocated_zone, buffer);
740059            if (status < 0)
740060              {
740061                //
740062                // Cannot overwrite the zone. The variable ‘errno’ is
740063                // set by ‘zone_write()’.
740064                //
740065                return ((zno_t) -1);
740066              }
740067            //
740068            // The zone is allocated and cleared: save the inode.
740069            //
740070            inode->direct[fzone] = allocated_zone;
740071            inode->changed = 1;
740072            status = inode_save (inode);
740073            if (status != 0)
740074              {
740075                //
740076                // Cannot save the inode. The variable ‘errno’ is
740077                // set ‘inode_save()’.
740078                //
740079                return ((zno_t) -1);
740080              }
740081          }
740082        //
740083        // The zone is there: return it.
740084        //
740085        return (inode->direct[fzone]);
740086      }
740087    if (fzone < 7 + indirect_zones)
740088      {
740089        //
740090        // 7 <= fzone <= (6 + indirect_zones)
740091        // The zone number is inside the single indirect zone
740092        // references: verify to have the indirect zone table.
```

```
740093        //
740094        if (inode->indirect1 == 0)
740095          {
740096            //
740097            // There is not such zone, but it is not an error.
740098            //
740099            if (!write)
740100              {
740101                return ((zno_t) 0);
740102              }
740103            //
740104            // The first level of indirection must be initialized.
740105            //
740106            allocated_zone = zone_alloc (inode->sb);
740107            if (allocated_zone == 0)
740108              {
740109                //
740110                // Cannot allocate the zone for the indirection table:
740111                // this is an error and the ‘errno’ value is produced
740112                // by ‘zone_alloc()’.
740113                //
740114                return ((zno_t) -1);
740115              }
740116            //
740117            // The zone for the indirection table is allocated:
740118            // clear the zone and save.
740119            //
740120            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740121            status = zone_write (inode->sb, allocated_zone, buffer);
740122            if (status < 0)
740123              {
740124                //
740125                // Cannot overwrite the zone. The variable ‘errno’ is
740126                // set by ‘zone_write()’.
740127                //
740128                return ((zno_t) -1);
740129              }
740130            //
740131            // The indirection table zone is allocated and cleared:
740132            // save the inode.
740133            //
740134            inode->indirect1 = allocated_zone;
740135            inode->changed = 1;
740136            status = inode_save (inode);
740137            if (status != 0)
740138              {
740139                //
740140                // Cannot save the inode. This is an error and the value
740141                // for ‘errno’ is produced by ‘inode_save()’.
740142                //
740143                return ((zno_t) -1);
740144              }
740145          }
740146        //
740147        // An indirect table is present inside the file system:
740148        // load it.
740149        //
740150        status = zone_read (inode->sb, inode->indirect1, zone_table);
740151        if (status != 0)
740152          {
740153            //
740154            // Cannot load the indirect table. This is an error and the
740155            // value for ‘errno’ is assigned by function ‘zone_read()’.
740156            //
740157            return ((zno_t) -1);
740158          }
740159        //
740160        // The indirect table was read. Calculate the index inside
740161        // the table, for the requested zone.
740162        //
740163        i0 = (fzone - 7);
740164        //
740165        // Check if the zone is to be allocated.
740166        //
740167        if (zone_table[i0] == 0)
740168          {
740169            //
740170            // There is not such zone, but it is not an error.
740171            //
740172            if (!write)
740173              {
740174                return ((zno_t) 0);
740175              }
740176            //
740177            // The zone must be allocated.
740178            //
740179            allocated_zone = zone_alloc (inode->sb);
740180            if (allocated_zone == 0)
740181              {
740182                //
740183                // There is no space for the zone allocation. The
740184                // variable ‘errno’ is already updated by
740185                // ‘zone_alloc()’.
740186                //
740187                return ((zno_t) -1);
740188              }
740189            //
740190            // The zone is allocated: clear the zone and save.
740191            //
740192            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740193            status = zone_write (inode->sb, allocated_zone, buffer);
```



```
740194            if (status < 0)
740195              {
740196                //
740197                // Cannot overwrite the zone. The variable ‘errno’ is
740198                // set by ‘zone_write()’.
740199                //
740200                return ((zno_t) -1);
740201              }
740202            //
740203            // The zone is allocated and cleared: update the indirect
740204            // zone table an save it. The inode is not modified,
740205            // because the indirect table is outside.
740206            //
740207            zone_table[i0] = allocated_zone;
740208            status = zone_write (inode->sb, inode->indirect1, zone_table);
740209            if (status != 0)
740210              {
740211                //
740212                // Cannot save the zone. The variable ‘errno’ is already
740213                // set by ‘zone_write()’.
740214                //
740215                return ((zno_t) -1);
740216              }
740217          }
740218        //
740219        // The zone is allocated.
740220        //
740221        return (zone_table[i0]);
740222      }
740223    else
740224      {
740225        //
740226        // (7 + indirect_zones) <= fzone
740227        // The zone number is inside the double indirect zone
740228        // references.
740229        // Verify to have the first level of second indirection.
740230        //
740231        if (inode->indirect2 == 0)
740232          {
740233            //
740234            // There is not such zone, but it is not an error.
740235            //
740236            if (!write)
740237              {
740238                return ((zno_t) 0);
740239              }
740240            //
740241            // The first level of second indirection must be
740242            // initialized.
740243            //
740244            allocated_zone = zone_alloc (inode->sb);
740245            if (allocated_zone == 0)
740246              {
740247                //
740248                // Cannot allocate the zone. The variable ‘errno’ is
740249                // set by ‘zone_alloc()’.
740250                //
740251                return ((zno_t) -1);
740252              }
740253            //
740254            // The zone for the indirection table is allocated:
740255            // clear the zone and save.
740256            //
740257            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740258            status = zone_write (inode->sb, allocated_zone, buffer);
740259            if (status < 0)
740260              {
740261                //
740262                // Cannot overwrite the zone. The variable ‘errno’ is
740263                // set by ‘zone_write()’.
740264                //
740265                return ((zno_t) -1);
740266              }
740267            //
740268            // The zone for the indirection table is allocated and
740269            // cleared: save the inode.
740270            //
740271            inode->indirect2 = allocated_zone;
740272            inode->changed = 1;
740273            status = inode_save (inode);
740274            if (status != 0)
740275              {
740276                //
740277                // Cannot save the inode. The variable ‘errno’ is
740278                // set by ‘inode_save()’.
740279                //
740280                return ((zno_t) -1);
740281              }
740282          }
740283        //
740284        // The first level of second indirection is present:
740285        // Read the second indirect table.
740286        //
740287        status = zone_read (inode->sb, inode->indirect2, zone_table);
740288        if (status != 0)
740289          {
740290            //
740291            // Cannot read the second indirect table. The variable
740292            // ‘errno’ is set by ‘zone_read()’.
740293            //
740294            return ((zno_t) -1);
```

```
740295          }
740296        //
740297        // The first double indirect table was read: calculate
740298        // indexes inside first and second level of table.
740299        //
740300        fzone -= 7;
740301        fzone -= indirect_zones;
740302        i1     = fzone / indirect_zones;
740303        i2     = fzone % indirect_zones;
740304        //
740305        // Verify to have a second level.
740306        //
740307        if (zone_table[i1] == 0)
740308          {
740309            //
740310            // There is not such zone, but it is not an error.
740311            //
740312            if (!write)
740313              {
740314                return ((zno_t) 0);
740315              }
740316            //
740317            // The second level must be initialized.
740318            //
740319            allocated_zone = zone_alloc (inode->sb);
740320            if (allocated_zone == 0)
740321              {
740322                //
740323                // Cannot allocate the zone. The variable ‘errno’ is set
740324                // by ‘zone_alloc()’.
740325                //
740326                return ((zno_t) -1);
740327              }
740328            //
740329            // The zone for the indirection table is allocated:
740330            // clear the zone and save.
740331            //
740332            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740333            status = zone_write (inode->sb, allocated_zone, buffer);
740334            if (status < 0)
740335              {
740336                //
740337                // Cannot overwrite the zone. The variable ‘errno’ is
740338                // set by ‘zone_write()’.
740339                //
740340                return ((zno_t) -1);
740341              }
740342            //
740343            // Update the first level index and save it.
740344            //
740345            zone_table[i1] = allocated_zone;
740346            status = zone_write (inode->sb, inode->indirect2, zone_table);
740347            if (status != 0)
740348              {
740349                //
740350                // Cannot write the zone. The variable ‘errno’ is set
740351                // by ‘zone_write()’.
740352                //
740353                return ((zno_t) -1);
740354              }
740355          }
740356        //
740357        // The second level can be read, overwriting the array
740358        // ‘zone_table[]’. The zone number for the second level
740359        // indirection table is saved inside ‘zone_second’, before
740360        // overwriting the array.
740361        //
740362        zone_second = zone_table[i1];
740363        status = zone_read (inode->sb, zone_second, zone_table);
740364        if (status != 0)
740365          {
740366            //
740367            // Cannot read the second level indirect table. The variable
740368            // ‘errno’ is set by ‘zone_read()’.
740369            //
740370            return ((zno_t) -1);
740371          }
740372        //
740373        // The second level was read and ‘zone_table[]’ is now
740374        // such second one: check if the zone is to be allocated.
740375        //
740376        if (zone_table[i2] == 0)
740377          {
740378            //
740379            // There is not such zone, but it is not an error.
740380            //
740381            if (!write)
740382              {
740383                return ((zno_t) 0);
740384              }
740385            //
740386            // Must be allocated.
740387            //
740388            allocated_zone = zone_alloc (inode->sb);
740389            if (allocated_zone == 0)
740390              {
740391                //
740392                // Cannot allocate the zone. The variable ‘errno’ is set
740393                // by ‘zone_alloc()’.
740394                //
740395                return ((zno_t) -1);
```



```
740396              }
740397            //
740398            // The zone is allocated: clear the zone and save.
740399            //
740400            memset (buffer, 0, SB_MAX_ZONE_SIZE);
740401            status = zone_write (inode->sb, allocated_zone, buffer);
740402            if (status < 0)
740403              {
740404                //
740405                // Cannot overwrite the zone. The variable ‘errno’ is
740406                // set by ‘zone_write()’.
740407                //
740408                return ((zno_t) -1);
740409              }
740410            //
740411            // The zone was allocated and cleared: update the indirect
740412            // zone table an save it. The inode is not modified, because
740413            // the indirect table is outside.
740414            //
740415            zone_table[i2] = allocated_zone;
740416            status = zone_write (inode->sb, zone_second, zone_table);
740417            if (status != 0)
740418              {
740419                //
740420                // Cannot write the zone. The variable ‘errno’ is set
740421                // by ‘zone_write()’.
740422                //
740423                return ((zno_t) -1);
740424              }
740425          }
740426        //
740427        // The zone is there: return the zone number.
740428        //
740429        return (zone_table[i2]);
740430      }
740431  }
```

kernel/fs/path_chdir.c

«

Si veda la sezione i159.3.30.

```
750001  #include <kernel/fs.h>
750002  #include <errno.h>
750003  #include <kernel/proc.h>
750004  //------------------------------------------------------------------------
750005  int
750006  path_chdir (pid_t pid, const char *path)
750007  {
750008      proc_t   *ps;
750009      inode_t  *inode_directory;
750010      int       status;
750011      char      path_directory[PATH_MAX];
750012      //
750013      // Get process.
750014      //
750015      ps = proc_reference (pid);
750016      //
750017      // The full directory path is needed.
750018      //
750019      status = path_full (path, ps->path_cwd, path_directory);
750020      if (status < 0)
750021        {
750022          return (-1);
750023        }
750024      //
750025      // Try to load the new directory inode.
750026      //
750027      inode_directory = path_inode (pid, path_directory);
750028      if (inode_directory == NULL)
750029        {
750030          //
750031          // Cannot access the directory: it does not exists or
750032          // permissions are not sufficient. Variable ‘errno’ is set by
750033          // function ‘inode_directory()’.
750034          //
750035          errset (errno);
750036          return (-1);
750037        }
750038      //
750039      // Inode loaded: release the old directory and set the new one.
750040      //
750041      inode_put (ps->inode_cwd);
750042      //
750043      ps->inode_cwd = inode_directory;
750044      strncpy (ps->path_cwd, path_directory, PATH_MAX);
750045      //
750046      // Return.
750047      //
750048      return (0);
750049  }
```

kernel/fs/path_chmod.c

«

Si veda la sezione i159.3.31.

```
760001  #include <kernel/fs.h>
760002  #include <errno.h>
760003  #include <kernel/proc.h>
760004  //------------------------------------------------------------------------
760005  int
```

```
760006  path_chmod (pid_t pid, const char *path, mode_t mode)
760007  {
760008      proc_t   *ps;
760009      inode_t  *inode;
760010      //
760011      // Get process.
760012      //
760013      ps = proc_reference (pid);
760014      //
760015      // Try to load the file inode.
760016      //
760017      inode = path_inode (pid, path);
760018      if (inode == NULL)
760019        {
760020          //
760021          // Cannot access the file: it does not exists or permissions are
760022          // not sufficient. Variable ‘errno’ is set by function
760023          // ‘inode_directory()’.
760024          //
760025          return (-1);
760026        }
760027      //
760028      // Verify to be root or to be the owner.
760029      //
760030      if (ps->euid != 0 && ps->euid != inode->uid)
760031        {
760032          errset (EACCES);        // Permission denied.
760033          return (-1);
760034        }
760035      //
760036      // Update the mode: the file type is kept and the
760037      // rest is taken form the parameter ‘mode’.
760038      //
760039      inode->mode = (S_IFMT & inode->mode) | (~S_IFMT & mode);
760040      //
760041      // Save and release the inode.
760042      //
760043      inode->changed = 1;
760044      inode_save (inode);
760045      inode_put (inode);
760046      //
760047      // Return.
760048      //
760049      return (0);
760050  }
```

kernel/fs/path_chown.c

«

Si veda la sezione i159.3.32.

```
770001  #include <kernel/fs.h>
770002  #include <errno.h>
770003  #include <kernel/proc.h>
770004  //-------------------------------------------------------------------------
770005  int
770006  path_chown (pid_t pid, const char *path, uid_t uid, gid_t gid)
770007  {
770008      proc_t   *ps;
770009      inode_t  *inode;
770010      //
770011      // Get process.
770012      //
770013      ps = proc_reference (pid);
770014      //
770015      // Must be root, as the ability to change group is not considered.
770016      //
770017      if (ps->euid != 0)
770018        {
770019          errset (EPERM);                 // Operation not permitted.
770020          return (-1);
770021        }
770022      //
770023      // Try to load the file inode.
770024      //
770025      inode = path_inode (pid, path);
770026      if (inode == NULL)
770027        {
770028          //
770029          // Cannot access the file: it does not exists or permissions are
770030          // not sufficient. Variable ‘errno’ is set by function
770031          // ‘inode_directory()’.
770032          //
770033          return (-1);
770034        }
770035      //
770036      // Update the owner and group.
770037      //
770038      if (uid != -1)
770039        {
770040          inode->uid     = uid;
770041          inode->changed = 1;
770042        }
770043      if (gid != -1)
770044        {
770045          inode->gid     = gid;
770046          inode->changed = 1;
770047        }
770048      //
770049      // Save and release the inode.
770050      //
770051      inode_save (inode);
```



```
770052      inode_put (inode);
770053      //
770054      // Return.
770055      //
770056      return (0);
770057  }
```

kernel/fs/path_device.c

«

Si veda la sezione i159.3.33.

```
780001  #include <kernel/fs.h>
780002  #include <errno.h>
780003  #include <kernel/proc.h>
780004  //-------------------------------------------------------------------------
780005  dev_t
780006  path_device (pid_t pid, const char *path)
780007  {
780008      proc_t   *ps;
780009      inode_t  *inode;
780010      dev_t     device;
780011      //
780012      // Get process.
780013      //
780014      ps = proc_reference (pid);
780015      //
780016      inode = path_inode (pid, path);
780017      if (inode == NULL)
780018        {
780019          errset (errno);
780020          return ((dev_t) -1);
780021        }
780022      //
780023      if (!(S_ISBLK (inode->mode) || S_ISCHR (inode->mode)))
780024        {
780025          errset (ENODEV);                // No such device.
780026          inode_put (inode);
780027          return ((dev_t) -1);
780028        }
780029      //
780030      device = inode->direct[0];
780031      inode_put (inode);
780032      return (device);
780033  }
```

kernel/fs/path_fix.c

«

Si veda la sezione i159.3.34.

```
790001  #include <kernel/fs.h>
790002  #include <errno.h>
790003  #include <kernel/proc.h>
790004  //-------------------------------------------------------------------------
790005  int
790006  path_fix (char *path)
790007  {
790008      char    new_path[PATH_MAX];
790009      char   *token[PATH_MAX/4];
790010      int     t;                          // Token index.
790011      int     token_size;                 // Token array effective size.
790012      int     comp;                       // String compare return value.
790013      size_t  path_size;                  // Path string size.
790014      //
790015      // Initialize token search.
790016      //
790017      token[0] = strtok (path, "/");
790018      //
790019      // Scan tokens.
790020      //
790021      for (t = 0;
790022           t < PATH_MAX/4 && token[t] != NULL;
790023           t++, token[t] = strtok (NULL, "/"))
790024        {
790025          //
790026          // If current token is ‘.’, just ignore it.
790027          //
790028          comp = strcmp (token[t], ".");
790029          if (comp == 0)
790030            {
790031              t--;
790032            }
790033          //
790034          // If current token is ‘..’, remove previous token,
790035          // if there is one.
790036          //
790037          comp = strcmp (token[t], "..");
790038          if (comp == 0)
790039            {
790040              if (t > 0)
790041                {
790042                  t -= 2;
790043                }
790044              else
790045                {
790046                  t = -1;
790047                }
790048            }
790049          //
790050          // ‘t’ will be incremented and another token will be
790051          // found.
```

```
790052          //
790053        }
790054      //
790055      // Save the token array effective size.
790056      //
790057      token_size = t;
790058      //
790059      // Initialize the new path string.
790060      //
790061      new_path[0] = '\0';
790062      //
790063      // Build the new path string.
790064      //
790065      if (token_size > 0)
790066        {
790067          for (t = 0; t < token_size; t++)
790068            {
790069              path_size = strlen (new_path);
790070              strncat (new_path, "/", 2);
790071              strncat (new_path, token[t], PATH_MAX - path_size - 1);
790072            }
790073        }
790074      else
790075        {
790076          strncat (new_path, "/", 2);
790077        }
790078      //
790079      // Copy the new path into the original string.
790080      //
790081      strncpy (path, new_path, PATH_MAX);
790082      //
790083      // Return.
790084      //
790085      return (0);
790086  }
```

kernel/fs/path_full.c

«

Si veda la sezione i159.3.35.

```
800001  #include <kernel/fs.h>
800002  #include <errno.h>
800003  #include <kernel/proc.h>
800004  //-------------------------------------------------------------------------
800005  int
800006  path_full (const char *path, const char *path_cwd, char *full_path)
800007  {
800008      unsigned int path_size;
800009      //
800010      // Check some arguments.
800011      //
800012      if (path == NULL || strlen (path) == 0 || full_path == NULL)
800013        {
800014          errset (EINVAL);                // Invalid argument.
800015          return (-1);
800016        }
800017      //
800018      // The main path and the receiving one are right.
800019      // Now arrange to get a full path name.
800020      //
800021      if (path[0] == '/')
800022        {
800023          strncpy (full_path, path, PATH_MAX);
800024          full_path[PATH_MAX-1] = 0;
800025        }
800026      else
800027        {
800028          if (path_cwd == NULL || strlen (path_cwd) == 0)
800029            {
800030              errset (EINVAL);            // Invalid argument.
800031              return (-1);
800032            }
800033          strncpy (full_path, path_cwd, PATH_MAX);
800034          path_size = strlen (full_path);
800035          strncat (full_path, "/", (PATH_MAX - path_size));
800036          path_size = strlen (full_path);
800037          strncat (full_path, path, (PATH_MAX - path_size));
800038        }
800039      //
800040      // Fix path name so that it has no '..', '.', and no
800041      // multiple '/'.
800042      //
800043      path_fix (full_path);
800044      //
800045      // Return.
800046      //
800047      return (0);
800048  }
```

kernel/fs/path_inode.c

«

Si veda la sezione i159.3.36.

```
810001  #include <kernel/fs.h>
810002  #include <errno.h>
810003  #include <kernel/proc.h>
810004  #include <kernel/k_libc.h>
810005  //-------------------------------------------------------------------------
810006  #define DIRECTORY_BUFFER_SIZE (SB_MAX_ZONE_SIZE/16)
810007  //-------------------------------------------------------------------------
```



```
810008  inode_t *
810009  path_inode (pid_t pid, const char *path)
810010  {
810011      proc_t     *ps;
810012      inode_t    *inode;
810013      dev_t       device;
810014      char        full_path[PATH_MAX];
810015      char       *name;
810016      char       *next;
810017      directory_t dir[DIRECTORY_BUFFER_SIZE];
810018      char        dir_name[NAME_MAX+1];
810019      off_t       offset_dir;
810020      ssize_t     size_read;
810021      size_t      dir_size_read;
810022      ssize_t     size_to_read;
810023      int         comp;
810024      int         d;              // Directory index;
810025      int         status;         // inode_check() return status.
810026      //
810027      // Get process.
810028      //
810029      ps = proc_reference (pid);
810030      //
810031      // Arrange to get a packed full path name.
810032      //
810033      path_full (path, ps->path_cwd, full_path);
810034      //
810035      // Get the root file system inode.
810036      //
810037      inode = inode_get ((dev_t) 0, 1);
810038      if (inode == NULL)
810039        {
810040          errset (errno);
810041          return (NULL);
810042        }
810043      //
810044      // Save the device number.
810045      //
810046      device = inode->sb->device;
810047      //
810048      // Variable 'inode' already points to the root file system inode:
810049      // It must be a directory!
810050      //
810051      status = inode_check (inode, S_IFDIR, 1, ps->euid);
810052      if (status != 0)
810053        {
810054          //
810055          // Variable 'errno' should be set by inode_check().
810056          //
810057          errset (errno);
810058          inode_put (inode);
810059          return (NULL);
810060        }
810061      //
810062      // Initialize string scan: find the first path token, after the
810063      // first '/'.
810064      //
810065      name = strtok (full_path, "/");
810066      //
810067      // If the original full path is just '/' the variable 'name'
810068      // appears as a null pointer, and the variable 'inode' is already
810069      // what we are looking for.
810070      //
810071      if (name == NULL)
810072        {
810073          return (inode);
810074        }
810075      //
810076      // There is at least a name after '/' inside the original full
810077      // path. A scan is going to start: the original value for variable
810078      // 'inode' is a pointer to the root directory inode.
810079      //
810080      for (;;)
810081        {
810082          //
810083          // Find next token.
810084          //
810085          next = strtok (NULL, "/");
810086          //
810087          // Read the directory from the current inode.
810088          //
810089          for (offset_dir=0; ; offset_dir += size_read)
810090            {
810091              size_to_read = DIRECTORY_BUFFER_SIZE;
810092              //
810093              if ((offset_dir + size_to_read) > inode->size)
810094                {
810095                  size_to_read = inode->size - offset_dir;
810096                }
810097              //
810098              size_read = inode_file_read (inode, offset_dir, dir,
810099                                           size_to_read, NULL);
810100              //
810101              // The size read must be a multiple of 16.
810102              //
810103              size_read = ((size_read / 16) * 16);
810104              //
810105              // Check anyway if it is zero.
810106              //
810107              if (size_read == 0)
810108                {
```

```
                //
                // The directory is ended: release the inode and return.
                //
                inode_put (inode);
                errset (ENOENT);           // No such file or directory.
                return (NULL);
              }
            //
            // Calculate how many directory items we have read.
            //
            dir_size_read = size_read / 16;
            //
            // Scan the directory to find the current name.
            //
            for (d = 0; d < dir_size_read; d++)
              {
                //
                // Ensure to have a null terminated string for
                // the name found.
                //
                memcpy (dir_name, dir[d].name, (size_t) NAME_MAX);
                dir_name[NAME_MAX] = 0;
                //
                comp = strcmp (name, dir_name);
                if (comp == 0 && dir[d].ino != 0)
                  {
                    //
                    // Found the name and verified that it has a link to
                    // a inode. Now release the directory inode.
                    //
                    inode_put (inode);
                    //
                    // Get next inode and break the loop.
                    //
                    inode = inode_get (device, dir[d].ino);
                    break;
                  }
              }
            //
            // If index 'd' is in a valid range, the name was found.
            //
            if (d < dir_size_read)
              {
                //
                // The name was found.
                //
                break;
              }
          }
        //
        // If the function is still working, a file or a directory
        // was found: see if there is another name after this one
        // to look for. If there isn't, just break the loop.
        //
        if (next == NULL)
          {
            //
            // As no other tokens are to be found, break the loop.
            //
            break;
          }
        //
        // As there is another name after the current one,
        // the current file must be a directory.
        //
        status = inode_check (inode, S_IFDIR, 1, ps->euid);
        if (status != 0)
          {
            //
            // Variable 'errno' is set by 'inode_check()'.
            //
            errset (errno);
            inode_put (inode);
            return (NULL);
         }
        //
        // The inode is a directory and the user has the necessary
        // permissions: check if it is a mount point and go to the
        // new device root directory if necessary.
        //
        if (inode->sb_attached != NULL)
          {
            //
            // Must find the root directory for the new device, and
            // then go to that inode.
            //
            device = inode->sb_attached->device;
            inode_put (inode);
            inode = inode_get (device, 1);
            status = inode_check (inode, S_IFDIR, 1, ps->euid);
            if (status != 0)
              {
                inode_put (inode);
                return (NULL);
              }
          }
        //
        // As a directory was found, and another token follows it,
        // must continue the token scan.
        //
        name = next;
```



```
      }
    //
    // Current inode found is the file represented by the requested
    // path.
    //
    return (inode);
}
```

kernel/fs/path_inode_link.c

«

Si veda la sezione i159.3.37.

```
#include <kernel/fs.h>
#include <errno.h>
#include <kernel/proc.h>
#include <libgen.h>
//-----------------------------------------------------------------------
inode_t *
path_inode_link (pid_t pid, const char *path, inode_t *inode,
                 mode_t mode)
{
    proc_t      *ps;
    char         buffer[SB_MAX_ZONE_SIZE];
    off_t        start;
    int          d;       // Directory index.
    ssize_t      size_read;
    ssize_t      size_written;
    directory_t *dir = (directory_t *) buffer;
    char         path_copy1[PATH_MAX];
    char         path_copy2[PATH_MAX];
    char        *path_directory;
    char        *path_name;
    inode_t     *inode_directory;
    inode_t     *inode_new;
    dev_t        device;
    int          status;
    //
    // Check arguments.
    //
    if (path == NULL || strlen (path) == 0)
      {
        errset (EINVAL);        // Invalid argument:
        return (NULL);          // the path is mandatory.
      }
    //
    if (inode == NULL && mode == 0)
      {
        errset (EINVAL);        // Invalid argument: if the inode is to
        return (NULL);          // be created, the mode is mandatory.
      }
    //
    if (inode != NULL)
      {
        if (mode != 0)
          {
            errset (EINVAL);    // Invalid argument: if the inode is
            return (NULL);      // already present, the creation mode
          }                     // must not be given.
        if (S_ISDIR (inode->mode))
          {
            errset (EPERM);     // Operation not permitted.
            return (NULL);      // Refuse to link directory.
          }
        if (inode->links >= LINK_MAX)
          {
            errset (EMLINK);            // Too many links.
            return (NULL);
          }
      }
    //
    // Get process.
    //
    ps = proc_reference (pid);
    //
    // If the destination path already exists, the link cannot be made.
    // It does not matter if the inode is known or not.
    //
    inode_new = path_inode ((uid_t) 0, path);
    if (inode_new != NULL)
      {
        //
        // A file already exists with the same name.
        //
        inode_put (inode_new);
        errset (EEXIST);                // File exists.
        return (NULL);
      }
    //
    // At this point, 'inode_new' is 'NULL'.
    // Copy the source path inside the directory path and name arrays.
    //
    strncpy (path_copy1, path, PATH_MAX);
    strncpy (path_copy2, path, PATH_MAX);
    //
    // Reduce to directory name and find the last name.
    //
    path_directory = dirname  (path_copy1);
    path_name      = basename (path_copy2);
    if (strlen (path_directory) == 0 || strlen (path_name) == 0)
      {
        errset (EACCES);        // Permission denied: maybe the
```

```
820090                                  // original path is the root directory
820091                                  // and cannot find a previous directory.
820092            return (NULL);
820093          }
820094        //
820095        // Get the directory inode.
820096        //
820097        inode_directory = path_inode (pid, path_directory);
820098        if (inode_directory == NULL)
820099          {
820100            errset (errno);
820101            return (NULL);
820102          }
820103        //
820104        // Check if something is mounted on it.
820105        //
820106        if (inode_directory->sb_attached != NULL)
820107          {
820108            //
820109            // Must select the right directory.
820110            //
820111            device = inode_directory->sb_attached->device;
820112            inode_put (inode_directory);
820113            inode_directory = inode_get (device, 1);
820114            if (inode_directory == NULL)
820115              {
820116                return (NULL);
820117              }
820118          }
820119        //
820120        // If the inode to link is known, check if the selected directory
820121        // has the same super block than the inode to link.
820122        //
820123        if (inode != NULL && inode_directory->sb != inode->sb)
820124          {
820125            inode_put (inode_directory);
820126            errset (ENOENT);        // No such file or directory.
820127            return (NULL);
820128          }
820129        //
820130        // Check if write is allowed for the file system.
820131        //
820132        if (inode_directory->sb->options & MOUNT_RO)
820133          {
820134            inode_put (inode_directory);
820135            errset (EROFS);         // Read-only file system.
820136            return (NULL);
820137          }
820138        //
820139        // Verify access permissions for the directory. The number "3" means
820140        // that the user must have access permission and write permission:
820141        // "-wx" == 2+1 == 3.
820142        //
820143        status = inode_check (inode_directory, S_IFDIR, 3, ps->euid);
820144        if (status != 0)
820145          {
820146            inode_put (inode_directory);
820147            return (NULL);
820148          }
820149        //
820150        // If the inode to link was not specified, it must be created.
820151        // From now on, the inode is referenced with the variable
820152        // 'inode_new'.
820153        //
820154        inode_new = inode;
820155        //
820156        if (inode_new == NULL)
820157          {
820158            inode_new = inode_alloc (inode_directory->sb->device, mode,
820159                                     ps->euid);
820160            if (inode_new == NULL)
820161              {
820162                //
820163                // The inode allocation failed, so, also the directory
820164                // must be released, before return.
820165                //
820166                inode_put (inode_directory);
820167                return (NULL);
820168              }
820169          }
820170        //
820171        // Read the directory content and try to add the new item.
820172        //
820173        for (start  = 0;
820174             start <  inode_directory->size;
820175             start += inode_directory->sb->blksize)
820176          {
820177            size_read = inode_file_read (inode_directory, start, buffer,
820178                                         inode_directory->sb->blksize,
820179                                         NULL);
820180            if (size_read < sizeof (directory_t))
820181              {
820182                break;
820183              }
820184            //
820185            // Scan the directory portion just read, for an unused item.
820186            //
820187            dir = (directory_t *) buffer;
820188            for (d = 0; d < size_read; d += (sizeof (directory_t)), dir++)
820189              {
820190                if (dir->ino == 0)
```



```
820191                  {
820192                    //
820193                    // Found an empty directory item: link the inode.
820194                    //
820195                    dir->ino = inode_new->ino;
820196                    strncpy (dir->name, path_name, NAME_MAX);
820197                    inode_new->links++;
820198                    inode_new->changed = 1;
820199                    //
820200                    // Update the directory inside the file system.
820201                    //
820202                    size_written = inode_file_write (inode_directory, start,
820203                                                     buffer, size_read);
820204                    if (size_written != size_read)
820205                      {
820206                        //
820207                        // Write problem: release the directory and return.
820208                        //
820209                        inode_put (inode_directory);
820210                        errset (EUNKNOWN);
820211                        return (NULL);
820212                      }
820213                    //
820214                    // Save the new inode, release the directory and return
820215                    // the linked inode.
820216                    //
820217                    inode_save (inode_new);
820218                    inode_put (inode_directory);
820219                    return (inode_new);
820220                  }
820221              }
820222          }
820223        //
820224        // The directory don't have a free item and one must be appended.
820225        //
820226        dir   = (directory_t *) buffer;
820227        start = inode_directory->size;
820228        //
820229        // Prepare the buffer with the link.
820230        //
820231        dir->ino = inode_new->ino;
820232        strncpy (dir->name, path_name, NAME_MAX);
820233        inode_new->links++;
820234        inode_new->changed = 1;
820235        //
820236        // Append the buffer to the directory.
820237        //
820238        size_written = inode_file_write (inode_directory, start, buffer,
820239                                         (sizeof (directory_t)));
820240        if (size_written != (sizeof (directory_t)))
820241          {
820242            //
820243            // Problem updating the directory: release it and return.
820244            //
820245            inode_put (inode_directory);
820246            errset (EUNKNOWN);
820247            return (NULL);
820248          }
820249        //
820250        // Close access to the directory inode and save the other inode,
820251        // with updated link count.
820252        //
820253        inode_put (inode_directory);
820254        inode_save (inode_new);
820255        //
820256        // Return successfully.
820257        //
820258        return (inode_new);
820259    }
```

kernel/fs/path_link.c

«

Si veda la sezione i159.3.38.

```
830001  #include <kernel/fs.h>
830002  #include <errno.h>
830003  #include <kernel/proc.h>
830004  //----------------------------------------------------------------------
830005  int
830006  path_link (pid_t pid, const char *path_old, const char *path_new)
830007  {
830008      proc_t             *ps;
830009      inode_t            *inode_old;
830010      inode_t            *inode_new;
830011      char                path_new_full[PATH_MAX];
830012      //
830013      // Get process.
830014      //
830015      ps = proc_reference (pid);
830016      //
830017      // Try to get the old path inode.
830018      //
830019      inode_old = path_inode (pid, path_old);
830020      if (inode_old == NULL)
830021        {
830022          //
830023          // Cannot get the inode: 'errno' is already set by
830024          // 'path_inode()'.
830025          //
830026          errset (errno);
830027          return (-1);
```

```
830028        }
830029      //
830030      // The inode is available and checks are done: arrange to get a
830031      // packed full path name and then the destination directory path.
830032      //
830033      path_full (path_new, ps->path_cwd, path_new_full);
830034      //
830035      //
830036      //
830037      inode_new = path_inode_link (pid, path_new_full, inode_old,
830038                                   (mode_t) 0);
830039      if (inode_new == NULL)
830040        {
830041          inode_put (inode_old);
830042          return (-1);
830043        }
830044      if (inode_new != inode_old)
830045        {
830046          inode_put (inode_new);
830047          inode_put (inode_old);
830048          errset (EUNKNOWN);              // Unknown error.
830049          return (-1);
830050        }
830051      //
830052      // Inode data is already updated by 'path_inode_link()': just put
830053      // it and return. Please note that only one is put, because it is
830054      // just the same of the other.
830055      //
830056      inode_put  (inode_new);
830057      return (0);
830058  }
```

kernel/fs/path_mkdir.c

«

Si veda la sezione i159.3.39.

```
840001  #include <kernel/fs.h>
840002  #include <errno.h>
840003  #include <kernel/proc.h>
840004  #include <libgen.h>
840005  #include <kernel/k_libc.h>
840006  //-------------------------------------------------------------------------
840007  int
840008  path_mkdir (pid_t pid, const char *path, mode_t mode)
840009  {
840010      proc_t   *ps;
840011      inode_t  *inode_directory;
840012      inode_t  *inode_parent;
840013      int       status;
840014      char      path_directory[PATH_MAX];
840015      char      path_copy[PATH_MAX];
840016      char     *path_parent;
840017      ssize_t   size_written;
840018      //
840019      struct {
840020          ino_t inode_1;
840021          char  name_1[NAME_MAX];
840022          ino_t inode_2;
840023          char  name_2[NAME_MAX];
840024      } directory;
840025      //
840026      // Get process.
840027      //
840028      ps = proc_reference (pid);
840029      //
840030      // Correct the mode with the umask.
840031      //
840032      mode &= ~ps->umask;
840033      //
840034      // Inside 'mode', the file type is fixed. No check is made.
840035      //
840036      mode &= 00777;
840037      mode |= S_IFDIR;
840038      //
840039      // The full path and the directory path is needed.
840040      //
840041      status = path_full (path, ps->path_cwd, path_directory);
840042      if (status < 0)
840043        {
840044          return (-1);
840045        }
840046      strncpy (path_copy, path_directory, PATH_MAX);
840047      path_copy[PATH_MAX-1] = 0;
840048      path_parent = dirname (path_copy);
840049      //
840050      // Check if something already exists with the same name. The scan
840051      // is done with kernel privileges.
840052      //
840053      inode_directory = path_inode ((uid_t) 0, path_directory);
840054      if (inode_directory != NULL)
840055        {
840056          //
840057          // The file already exists. Put inode and return an error.
840058          //
840059          inode_put (inode_directory);
840060          errset (EEXIST);                // File exists.
840061          return (-1);
840062        }
840063      //
840064      // Try to locate the directory that should contain this one.
840065      //
```



```
840066      inode_parent = path_inode (pid, path_parent);
840067      if (inode_parent == NULL)
840068        {
840069          //
840070          // Cannot locate the directory: return an error. The variable
840071          // 'errno' should already be set by 'path_inode()'.
840072          //
840073          errset (errno);
840074          return (-1);
840075        }
840076      //
840077      // Try to create the node: should fail if the user does not have
840078      // enough permissions.
840079      //
840080      inode_directory = path_inode_link (pid, path_directory, NULL,
840081                                         mode);
840082      if (inode_directory == NULL)
840083        {
840084          //
840085          // Sorry: cannot create the inode! The variable 'errno' should
840086          // already be set by 'path_inode_link()'.
840087          //
840088          errset (errno);
840089          return (-1);
840090        }
840091      //
840092      // Fill records for '.' and '..'.
840093      //
840094      directory.inode_1 = inode_directory->ino;
840095      strncpy (directory.name_1, ".",  (size_t) 3);
840096      directory.inode_2 = inode_parent->ino;
840097      strncpy (directory.name_2, "..", (size_t) 3);
840098      //
840099      // Write data.
840100      //
840101      size_written = inode_file_write (inode_directory, (off_t) 0,
840102                                       &directory, (sizeof directory));
840103      if (size_written != (sizeof directory))
840104        {
840105          return (-1);
840106        }
840107      //
840108      // Fix directory inode links.
840109      //
840110      inode_directory->links   = 2;
840111      inode_directory->time    = k_time (NULL);
840112      inode_directory->changed = 1;
840113      //
840114      // Fix parent directory inode links.
840115      //
840116      inode_parent->links++;
840117      inode_parent->time       = k_time (NULL);
840118      inode_parent->changed    = 1;
840119      //
840120      // Save and put the inodes.
840121      //
840122      inode_save (inode_parent);
840123      inode_save (inode_directory);
840124      inode_put  (inode_parent);
840125      inode_put  (inode_directory);
840126      //
840127      // Return.
840128      //
840129      return (0);
840130  }
```

kernel/fs/path_mknod.c

«

Si veda la sezione i159.3.40.

```
850001  #include <kernel/fs.h>
850002  #include <errno.h>
850003  #include <kernel/proc.h>
850004  //-------------------------------------------------------------------------
850005  int
850006  path_mknod (pid_t pid, const char *path, mode_t mode, dev_t device)
850007  {
850008      proc_t   *ps;
850009      inode_t  *inode;
850010      char      full_path[PATH_MAX];
850011      //
850012      // Get process.
850013      //
850014      ps = proc_reference (pid);
850015      //
850016      // Correct the mode with the umask.
850017      //
850018      mode &= ~ps->umask;
850019      //
850020      // Currently must be root for any kind of node to be created.
850021      //
850022      if (ps->uid != 0)
850023        {
850024          errset (EPERM);                 // Operation not permitted.
850025          return (-1);
850026        }
850027      //
850028      // Check the type of node requested.
850029      //
850030      if (!(S_ISBLK (mode) ||
850031            S_ISCHR (mode) ||
```

```
850032            S_ISREG (mode) ||
850033            S_ISDIR (mode)))
850034        {
850035          errset (EINVAL);                // Invalid argument.
850036          return (-1);
850037        }
850038      //
850039      // Check if something already exists with the same name.
850040      //
850041      inode = path_inode (pid, path);
850042      if (inode != NULL)
850043        {
850044          //
850045          // The file already exists. Put inode and return an error.
850046          //
850047          inode_put (inode);
850048          errset (EEXIST);                // File exists.
850049          return (-1);
850050        }
850051      //
850052      // Try to creat the node.
850053      //
850054      path_full (path, ps->path_cwd, full_path);
850055      inode = path_inode_link (pid, full_path, NULL, mode);
850056      if (inode == NULL)
850057        {
850058          //
850059          // Sorry: cannot create the inode!
850060          //
850061          return (-1);
850062        }
850063      //
850064      // Set the device number if necessary.
850065      //
850066      if (S_ISBLK (mode) || S_ISCHR (mode))
850067        {
850068          inode->direct[0] = device;
850069          inode->changed = 1;
850070        }
850071      //
850072      // Put the inode.
850073      //
850074      inode_put (inode);
850075      //
850076      // Return.
850077      //
850078      return (0);
850079  }
```

kernel/fs/path_mount.c

«

Si veda la sezione i159.3.41.

```
860001  #include <kernel/fs.h>
860002  #include <errno.h>
860003  #include <kernel/proc.h>
860004  //-----------------------------------------------------------------------
860005  int
860006  path_mount (pid_t pid, const char *path_dev, const char *path_mnt,
860007              int options)
860008  {
860009      proc_t   *ps;
860010      dev_t     device;        // Device to mount.
860011      inode_t  *inode_mnt;     // Directory mount point.
860012      void     *pstatus;
860013      //
860014      // Get process.
860015      //
860016      ps = proc_reference (pid);
860017      //
860018      // Verify to be the super user.
860019      //
860020      if (ps->euid != 0)
860021        {
860022          errset (EPERM);         // Operation not permitted.
860023          return (-1);
860024        }
860025      //
860026      device = path_device (pid, path_dev);
860027      if (device < 0)
860028        {
860029          return (-1);
860030        }
860031      //
860032      inode_mnt = path_inode (pid, path_mnt);
860033      if (inode_mnt == NULL)
860034        {
860035          return (-1);
860036        }
860037      if (!S_ISDIR (inode_mnt->mode))
860038        {
860039          inode_put (inode_mnt);
860040          errset (ENOTDIR);       // Not a directory.
860041          return (-1);
860042        }
860043      if (inode_mnt->sb_attached != NULL)
860044        {
860045          inode_put (inode_mnt);
860046          errset (EBUSY);         // Device or resource busy.
860047          return (-1);
860048        }
```



```
860049      //
860050      // All data is available.
860051      //
860052      pstatus = sb_mount (device, &inode_mnt, options);
860053      if (pstatus == NULL)
860054        {
860055          inode_put (inode_mnt);
860056          return (-1);
860057        }
860058      //
860059      return (0);
860060  }
```

kernel/fs/path_stat.c

«

Si veda la sezione i159.3.50.

```
870001  #include <kernel/fs.h>
870002  #include <errno.h>
870003  #include <kernel/proc.h>
870004  //-----------------------------------------------------------------------
870005  int
870006  path_stat (pid_t pid, const char *path, struct stat *buffer)
870007  {
870008      proc_t   *ps;
870009      inode_t  *inode;
870010      //
870011      // Get process.
870012      //
870013      ps = proc_reference (pid);
870014      //
870015      // Try to load the file inode.
870016      //
870017      inode = path_inode (pid, path);
870018      if (inode == NULL)
870019        {
870020          //
870021          // Cannot access the file: it does not exists or permissions are
870022          // not sufficient. Variable 'errno' is set by function
870023          // 'path_inode()'.
870024          //
870025          errset (errno);
870026          return (-1);
870027        }
870028      //
870029      // Inode loaded: update the buffer.
870030      //
870031      buffer->st_dev      = inode->sb->device;
870032      buffer->st_ino      = inode->ino;
870033      buffer->st_mode     = inode->mode;
870034      buffer->st_nlink    = inode->links;
870035      buffer->st_uid      = inode->uid;
870036      buffer->st_gid      = inode->gid;
870037      if (S_ISBLK (buffer->st_mode) || S_ISCHR (buffer->st_mode))
870038        {
870039          buffer->st_rdev = inode->direct[0];
870040        }
870041      else
870042        {
870043          buffer->st_rdev = 0;
870044        }
870045      buffer->st_size     = inode->size;
870046      buffer->st_atime    = inode->time;  // All times are the same for
870047      buffer->st_mtime    = inode->time;  // Minix 1 file system.
870048      buffer->st_ctime    = inode->time;  //
870049      buffer->st_blksize  = inode->sb->blksize;
870050      buffer->st_blocks   = inode->blkcnt;
870051      //
870052      // If the inode is a device special file, the 'st_rdev' value is
870053      // taken from the first direct zone (as of Minix 1 organization).
870054      //
870055      if (S_ISBLK(inode->mode) || S_ISCHR(inode->mode))
870056        {
870057          buffer->st_rdev = inode->direct[0];
870058        }
870059      else
870060        {
870061          buffer->st_rdev = 0;
870062        }
870063      //
870064      // Release the inode and return.
870065      //
870066      inode_put (inode);
870067      //
870068      // Return.
870069      //
870070      return (0);
870071  }
```

kernel/fs/path_umount.c

«

Si veda la sezione i159.3.41.

```
880001  #include <kernel/fs.h>
880002  #include <errno.h>
880003  #include <kernel/proc.h>
880004  //-----------------------------------------------------------------------
880005  int
880006  path_umount (pid_t pid, const char *path_mnt)
880007  {
```

```
880008      proc_t   *ps;
880009      dev_t     device;               // Device to mount.
880010      inode_t  *inode_mount_point;    // Original mount point.
880011      inode_t  *inode;                // Inode table.
880012      int       i;                    // Inode table index.
880013      //
880014      // Get process.
880015      //
880016      ps = proc_reference (pid);
880017      //
880018      // Verify to be the super user.
880019      //
880020      if (ps->euid != 0)
880021        {
880022          errset (EPERM);         // Operation not permitted.
880023          return (-1);
880024        }
880025      //
880026      // Get the directory mount point.
880027      //
880028      inode_mount_point = path_inode (pid, path_mnt);
880029      if (inode_mount_point == NULL)
880030        {
880031          errset (ENOENT);                // No such file or directory.
880032          return (-1);
880033        }
880034      //
880035      // Verify that the path is a directory.
880036      //
880037      if (!S_ISDIR (inode_mount_point->mode))
880038        {
880039          inode_put (inode_mount_point);
880040          errset (ENOTDIR);               // Not a directory.
880041          return (-1);
880042        }
880043      //
880044      // Verify that there is something attached.
880045      //
880046      device = inode_mount_point->sb_attached->device;
880047      if (device == 0)
880048        {
880049          //
880050          // There is nothing to unmount.
880051          //
880052          inode_put (inode_mount_point);
880053          errset (E_NOT_MOUNTED);         // Not mounted.
880054          return (-1);
880055        }
880056      //
880057      // Are there exactly two internal references? Let's explain:
880058      // the directory that act as mount point, should have one reference
880059      // because it is mounting something and another because it was just
880060      // opened again, a few lines above. If there are more references
880061      // it is wrong; if there are less, it is also wrong at this point.
880062      //
880063      if (inode_mount_point->references != 2)
880064        {
880065          inode_put (inode_mount_point);
880066          errset (EUNKNOWN);              // Unknown error.
880067          return (-1);
880068        }
880069      //
880070      // All data is available: find if there are open file inside
880071      // the file system to unmount. But first load the inode table
880072      // pointer.
880073      //
880074      inode = inode_reference ((dev_t) 0, (ino_t) 0);
880075      if (inode == NULL)
880076        {
880077          //
880078          // This error should not happen.
880079          //
880080          inode_put (inode_mount_point);
880081          errset (EUNKNOWN);              // Unknown error.
880082          return (-1);
880083        }
880084      //
880085      // Scan the inode table.
880086      //
880087      for (i = 0; i < INODE_MAX_SLOTS; i++)
880088        {
880089          if (inode[i].sb == inode_mount_point->sb_attached  &&
880090              inode[i].references > 0)
880091            {
880092              //
880093              // At least one file is open inside the super block to
880094              // release: cannot unmount.
880095              //
880096              inode_put (inode_mount_point);
880097              errset (EBUSY);             // Device or resource busy.
880098              return (-1);
880099            }
880100        }
880101      //
880102      // Can unmount: save and remove the super block memory;
880103      // clear the mount point reference and put inode.
880104      //
880105      inode_mount_point->sb_attached->changed          = 1;
880106      sb_save (inode_mount_point->sb_attached);
880107      //
880108      inode_mount_point->sb_attached->device           = 0;
```



```
880109      inode_mount_point->sb_attached->inode_mounted_on = NULL;
880110      inode_mount_point->sb_attached->blksize          = 0;
880111      inode_mount_point->sb_attached->options          = 0;
880112      //
880113      inode_mount_point->sb_attached                   = NULL;
880114      inode_mount_point->references                    = 0;
880115      inode_put (inode_mount_point);
880116      //
880117      inode_put (inode_mount_point);
880118      //
880119      return (0);
880120  }
```

kernel/fs/path_unlink.c

«

Si veda la sezione i159.3.44.

```
890001  #include <kernel/fs.h>
890002  #include <errno.h>
890003  #include <kernel/proc.h>
890004  #include <libgen.h>
890005  #include <kernel/k_libc.h>
890006  //------------------------------------------------------------------------
890007  int
890008  path_unlink (pid_t pid, const char *path)
890009  {
890010      proc_t      *ps;
890011      inode_t     *inode_unlink;
890012      inode_t     *inode_directory;
890013      char         path_unlink[PATH_MAX];
890014      char         path_copy[PATH_MAX];
890015      char        *path_directory;
890016      char        *name_unlink;
890017      dev_t        device;
890018      off_t        start;
890019      char         buffer[SB_MAX_ZONE_SIZE];
890020      directory_t *dir = (directory_t *) buffer;
890021      int          status;
890022      ssize_t      size_read;
890023      ssize_t      size_written;
890024      int          d;                     // Directory buffer index.
890025      //
890026      // Get process.
890027      //
890028      ps = proc_reference (pid);
890029      //
890030      // Get full paths.
890031      //
890032      path_full (path, ps->path_cwd, path_unlink);
890033      strncpy (path_copy, path_unlink, PATH_MAX);
890034      path_directory = dirname (path_copy);
890035      //
890036      // Get the inode to be unlinked.
890037      //
890038      inode_unlink = path_inode (pid, path_unlink);
890039      if (inode_unlink == NULL)
890040        {
890041          return (-1);
890042        }
890043      //
890044      // If it is a directory, verify that it is empty.
890045      //
890046      if (S_ISDIR (inode_unlink->mode))
890047        {
890048          if (!inode_dir_empty (inode_unlink))
890049            {
890050              inode_put (inode_unlink);
890051              errset (ENOTEMPTY);         // Directory not empty.
890052              return (-1);
890053            }
890054        }
890055      //
890056      // Get the inode of the directory containing it.
890057      //
890058      inode_directory = path_inode (pid, path_directory);
890059      if (inode_directory == NULL)
890060        {
890061          inode_put (inode_unlink);
890062          return (-1);
890063        }
890064      //
890065      // Check if something is mounted on the directory.
890066      //
890067      if (inode_directory->sb_attached != NULL)
890068        {
890069          //
890070          // Must select the right directory.
890071          //
890072          device = inode_directory->sb_attached->device;
890073          inode_put (inode_directory);
890074          inode_directory = inode_get (device, 1);
890075          if (inode_directory == NULL)
890076            {
890077              inode_put (inode_unlink);
890078              return (-1);
890079            }
890080        }
890081      //
890082      // Check if write is allowed for the file system.
890083      //
890084      if (inode_directory->sb->options & MOUNT_RO)
```

```
890085        {
890086          errset (EROFS);         // Read-only file system.
890087          return (-1);
890088        }
890089      //
890090      // Verify access permissions for the directory. The number "3" means
890091      // that the user must have access permission and write permission:
890092      // "-wx" == 2+1 == 3.
890093      //
890094      status = inode_check (inode_directory, S_IFDIR, 3, ps->uid);
890095      if (status != 0)
890096        {
890097          errset (EPERM);                 // Operation not permitted.
890098          inode_put (inode_unlink);
890099          inode_put (inode_directory);
890100          return (-1);
890101        }
890102      //
890103      // Get the base name to be unlinked: this will alter the
890104      // original path.
890105      //
890106      name_unlink = basename (path_unlink);
890107      //
890108      // Read the directory content and try to locate the item to unlink.
890109      //
890110      for (start  = 0;
890111           start <  inode_directory->size;
890112           start += inode_directory->sb->blksize)
890113        {
890114          size_read = inode_file_read (inode_directory, start, buffer,
890115                                       inode_directory->sb->blksize,
890116                                       NULL);
890117          if (size_read < sizeof (directory_t))
890118            {
890119              break;
890120            }
890121          //
890122          // Scan the directory portion just read, for the item to unlink.
890123          //
890124          dir = (directory_t *) buffer;
890125          //
890126          for (d = 0; d < size_read; d += (sizeof (directory_t)), dir++)
890127            {
890128              if (dir->ino != 0                                       &&
890129                  strncmp (dir->name, name_unlink, NAME_MAX) == 0)
890130                {
890131                  //
890132                  // Found the corresponding item: unlink the inode.
890133                  //
890134                  dir->ino     = 0;
890135                  //
890136                  // Update the directory inside the file system.
890137                  //
890138                  size_written = inode_file_write (inode_directory, start,
890139                                                   buffer, size_read);
890140                  if (size_written != size_read)
890141                    {
890142                      //
890143                      // Write problem: just tell.
890144                      //
890145                      k_printf ("kernel alert: directory write error!\n");
890146                    }
890147                  //
890148                  // Update directory inode and put inode. If the unlinked
890149                  // inode was a directory, the parent directory inode
890150                  // must reduce the file system link count.
890151                  //
890152                  if (S_ISDIR (inode_unlink->mode))
890153                    {
890154                      inode_directory->links--;
890155                    }
890156                  inode_directory->time = k_time (NULL);
890157                  inode_directory->changed = 1;
890158                  inode_put (inode_directory);
890159                  //
890160                  // Reduce link inside unlinked inode and put inode.
890161                  //
890162                  inode_unlink->links--;
890163                  inode_unlink->changed = 1;
890164                  inode_unlink->time = k_time (NULL);
890165                  inode_put (inode_unlink);
890166                  //
890167                  // Just return, as the work is done.
890168                  //
890169                  return (0);
890170                }
890171            }
890172        }
890173      //
890174      // At this point, it was not possible to unlink the file.
890175      //
890176      inode_put (inode_unlink);
890177      inode_put (inode_directory);
890178      errset (EUNKNOWN);          // Unknown error.
890179      return (-1);
890180  }
```


kernel/fs/sb_inode_status.c

«

Si veda la sezione i159.3.45.

```
900001  #include <kernel/fs.h>
900002  #include <errno.h>
900003  //-------------------------------------------------------------------------
900004  int
900005  sb_inode_status (sb_t *sb, ino_t ino)
900006  {
900007      int map_element;
900008      int map_bit;
900009      int map_mask;
900010      //
900011      // Check arguments.
900012      //
900013      if (ino == 0 || sb == NULL)
900014        {
900015          errset (EINVAL);        // Invalid argument.
900016          return (-1);
900017        }
900018      //
900019      //  Calculate the map element, the map bit and the map mask.
900020      //
900021      map_element = ino / 16;
900022      map_bit     = ino % 16;
900023      map_mask    = 1 << map_bit;
900024      //
900025      // Check the inode and return.
900026      //
900027      if (sb->map_inode[map_element] & map_mask)
900028        {
900029          return (1);     // True.
900030        }
900031      else
900032        {
900033          return (0);     // False.
900034        }
900035  }
```

kernel/fs/sb_mount.c

«

Si veda la sezione i159.3.46.

```
910001  #include <kernel/fs.h>
910002  #include <errno.h>
910003  #include <kernel/devices.h>
910004  //-------------------------------------------------------------------------
910005  sb_t *
910006  sb_mount (dev_t device, inode_t **inode_mnt, int options)
910007  {
910008      sb_t   *sb;
910009      ssize_t size_read;
910010      addr_t  start;
910011      int     m;
910012      size_t  size_sb;
910013      size_t  size_map;
910014      //
910015      // Find if it is already mounted.
910016      //
910017      sb = sb_reference (device);
910018      if (sb != NULL)
910019        {
910020          errset (EBUSY);         // Device or resource busy: device
910021          return (NULL);          // already mounted.
910022        }
910023      //
910024      // Find if '*inode_mnt' is already mounting something.
910025      //
910026      if (*inode_mnt != NULL && (*inode_mnt)->sb_attached != NULL)
910027        {
910028          errset (EBUSY);         // Device or resource busy: mount point
910029          return (NULL);          // already used.
910030        }
910031      //
910032      // The inode is not yet mounting anything, or it is new: find a free
910033      // slot inside the super block table.
910034      //
910035      sb = sb_reference ((dev_t) -1);
910036      if (sb == NULL)
910037        {
910038          errset (EBUSY);         // Device or resource busy:
910039          return (NULL);          // no free slots.
910040        }
910041      //
910042      // A free slot was found: the super block header must be loaded, but
910043      // before it is necessary to calculate the header size to be read.
910044      //
910045      size_sb = offsetof (sb_t, device);
910046      //
910047      // Then fix the starting point.
910048      //
910049      start  = 1024;                      // After boot block.
910050      //
910051      // Read the file system super block header.
910052      //
910053      size_read = dev_io ((pid_t) -1, device, DEV_READ, start, sb,
910054                          size_sb, NULL);
910055      if (size_read != size_sb)
910056        {
910057          errset (EIO);                   // I/O error.
```

```
910058          return (NULL);
910059        }
910060      //
910061      // Save some more data.
910062      //
910063      sb->device           =  device;
910064      sb->options          =  options;
910065      sb->inode_mounted_on = *inode_mnt;
910066      sb->blksize          =  (1024 << sb->log2_size_zone);
910067      //
910068      // Check if the super block data is valid.
910069      //
910070      if (sb->magic_number != 0x137F)
910071        {
910072          errset (ENODEV);        // No such device: unsupported
910073          sb->device = 0;         // file system type.
910074          return (NULL);
910075        }
910076      if (sb->map_inode_blocks > SB_MAX_INODE_BLOCKS)
910077        {
910078          errset (E_MAP_INODE_TOO_BIG);
910079          return (NULL);
910080        }
910081      if (sb->map_zone_blocks > SB_MAX_ZONE_BLOCKS)
910082        {
910083          errset (E_MAP_ZONE_TOO_BIG);
910084          return (NULL);
910085        }
910086      if (sb->blksize > SB_MAX_ZONE_SIZE)
910087        {
910088          errset (E_DATA_ZONE_TOO_BIG);
910089          return (NULL);
910090        }
910091      //
910092      // A right super block header was loaded from disk, now load the
910093      // super block inode bit map.
910094      //
910095      start  = 1024;                              // After boot block.
910096      start += 1024;                              // After super block.
910097      for (m = 0; m < SB_MAP_INODE_SIZE; m++)     //
910098        {                                         // Reset map in memory,
910099          sb->map_inode[m] = 0xFFFF;              // before loading.
910100        }                                         //
910101      size_map = sb->map_inode_blocks * 1024;
910102      size_read = dev_io ((pid_t) -1, sb->device, DEV_READ, start,
910103                          sb->map_inode, size_map, NULL);
910104      if (size_read != size_map)
910105        {
910106          errset (EIO);                   // I/O error.
910107          return (NULL);
910108        }
910109      //
910110      // Load the super block zone bit map.
910111      //
910112      start  = 1024;                              // After boot block.
910113      start += 1024;                              // After super block.
910114      start += (sb->map_inode_blocks * 1024);     // After inode bit map.
910115      for (m = 0; m < SB_MAP_ZONE_SIZE; m++)      //
910116        {                                         // Reset map in memory,
910117          sb->map_zone[m] = 0xFFFF;               // before loading.
910118        }                                         //
910119      size_map = sb->map_zone_blocks * 1024;
910120      size_read = dev_io ((pid_t) -1, sb->device, DEV_READ, start,
910121                          sb->map_zone, size_map, NULL);
910122      if (size_read != size_map)
910123        {
910124          errset (EIO);                   // I/O error.
910125          return (NULL);
910126        }
910127      //
910128      // Check the inode that should mount the super block. If
910129      // '*inode_mnt' is 'NULL', then it is meant to be the first mount of
910130      // the root file system. In such case, the inode must be loaded too,
910131      // and the value for '*inode_mnt' must be modified.
910132      //
910133      if (*inode_mnt == NULL)
910134        {
910135          *inode_mnt = inode_get (device, 1);
910136        }
910137      //
910138      // Check for a valid value.
910139      //
910140      if (*inode_mnt == NULL)
910141        {
910142          //
910143          // This is bad!
910144          //
910145          errset (EUNKNOWN);              // Unknown error.
910146          return (NULL);
910147        }
910148      //
910149      // A valid inode is available for the mount.
910150      //
910151      (*inode_mnt)->sb_attached  = sb;
910152      //
910153      // Return the super block pointer.
910154      //
910155      return (sb);
910156  }
```


kernel/fs/sb_reference.c

«

Si veda la sezione i159.3.47.

```
920001  #include <kernel/fs.h>
920002  #include <errno.h>
920003  //------------------------------------------------------------------------
920004  sb_t *
920005  sb_reference (dev_t device)
920006  {
920007      int s;                              // Slot index.
920008      //
920009      // If device is zero, a reference to the whole table is returned.
920010      //
920011      if (device == 0)
920012        {
920013          return (sb_table);
920014        }
920015      //
920016      // If device is ((dev_t) -1), a reference to a free slot is
920017      // returned.
920018      //
920019      if (device == ((dev_t) -1))
920020        {
920021          for (s = 0; s < SB_MAX_SLOTS; s++)
920022            {
920023              if (sb_table[s].device == 0)
920024                {
920025                  return (&sb_table[s]);
920026                }
920027            }
920028          return (NULL);
920029        }
920030      //
920031      // A device was selected: find the super block associated to it.
920032      //
920033      for (s = 0; s < SB_MAX_SLOTS; s++)
920034        {
920035          if (sb_table[s].device == device)
920036            {
920037              return (&sb_table[s]);
920038            }
920039        }
920040      //
920041      // The super block was not found.
920042      //
920043      return (NULL);
920044  }
```

kernel/fs/sb_save.c

«

Si veda la sezione i159.3.48.

```
930001  #include <kernel/fs.h>
930002  #include <errno.h>
930003  #include <kernel/devices.h>
930004  //------------------------------------------------------------------------
930005  int
930006  sb_save (sb_t *sb)
930007  {
930008      ssize_t size_written;
930009      addr_t  start;
930010      size_t  size_map;
930011      //
930012      // Check for valid argument.
930013      //
930014      if (sb == NULL)
930015        {
930016          errset (EINVAL);                        // Invalid argument.
930017          return (-1);
930018        }
930019      //
930020      // Check if the super block changed for some reason (only the
930021      // inode and the zone maps can change really).
930022      //
930023      if (!sb->changed)
930024        {
930025          //
930026          // Nothing to save.
930027          //
930028          return (0);
930029        }
930030      //
930031      // Something inside the super block changed: start the procedure to
930032      // save the inode map (recall that the super block header is not
930033      // saved, because it never changes).
930034      //
930035      start  = 1024;                              // After boot block.
930036      start += 1024;                              // After super block.
930037      size_map = sb->map_inode_blocks * 1024;
930038      size_written = dev_io ((pid_t) -1, sb->device, DEV_WRITE, start,
930039                  sb->map_inode, size_map, NULL);
930040      if (size_written != size_map)
930041        {
930042          //
930043          // Error writing the map.
930044          //
930045          errset (EIO);                   // I/O error.
930046          return (-1);
930047        }
930048      //
```

```
930049      // Start the procedure to save the zone map.
930050      //
930051      start  = 1024;                              // After boot block.
930052      start += 1024;                              // After super block.
930053      start += (sb->map_inode_blocks * 1024);     // After inode bit map.
930054      size_map = sb->map_zone_blocks * 1024;
930055      size_written = dev_io ((pid_t) -1, sb->device, DEV_WRITE, start,
930056                   sb->map_zone, size_map, NULL);
930057      if (size_written != size_map)
930058        {
930059          //
930060          // Error writing the map.
930061          //
930062          errset (EIO);                   // I/O error.
930063          return (-1);
930064        }
930065      //
930066      // Super block saved.
930067      //
930068      sb->changed = 0;
930069      //
930070      return (0);
930071  }
```

kernel/fs/sb_table.c

«

Si veda la sezione i159.3.47.

```
940001  #include <kernel/fs.h>
940002  //-------------------------------------------------------------------------
940003  sb_t    sb_table[SB_MAX_SLOTS];
```

kernel/fs/sb_zone_status.c

«

Si veda la sezione i159.3.45.

```
950001  #include <kernel/fs.h>
950002  #include <errno.h>
950003  //-------------------------------------------------------------------------
950004  int
950005  sb_zone_status (sb_t *sb, zno_t zone)
950006  {
950007      int map_element;
950008      int map_bit;
950009      int map_mask;
950010      //
950011      // Check arguments.
950012      //
950013      if (zone == 0 || sb == NULL)
950014        {
950015          errset (EINVAL);        // Invalid argument.
950016          return (-1);
950017        }
950018      //
950019      //  Calculate the map element, the map bit and the map mask.
950020      //
950021      map_element = zone / 16;
950022      map_bit     = zone % 16;
950023      map_mask    = 1 << map_bit;
950024      //
950025      // Check the zone and return.
950026      //
950027      if (sb->map_zone[map_element] & map_mask)
950028        {
950029          return (1);     // True.
950030        }
950031      else
950032        {
950033          return (0);     // False.
950034        }
950035  }
```

kernel/fs/zone_alloc.c

«

Si veda la sezione i159.3.51.

```
960001  #include <kernel/fs.h>
960002  #include <kernel/devices.h>
960003  #include <errno.h>
960004  //-------------------------------------------------------------------------
960005  zno_t
960006  zone_alloc (sb_t *sb)
960007  {
960008      int   m;                    // Index inside the inode map.
960009      int   map_element;
960010      int   map_bit;
960011      int   map_mask;
960012      zno_t zone;
960013      char  buffer[SB_MAX_ZONE_SIZE];
960014      int   status;
960015      //
960016      // Verify if write is allowed.
960017      //
960018      if (sb->options & MOUNT_RO)
960019        {
960020          errset (EROFS);         // Read-only file system.
960021          return ((zno_t) 0);
960022        }
```



```
960023      //
960024      // Write allowed: scan the zone map, to find a free zone.
960025      // If a free zone can be found, allocate it inside the map.
960026      // Index 'm' starts from one, because the first bit of the
960027      // map is reserved for a 'zero' data-zone that does not
960028      // exist: the second bit is for the real first data-zone.
960029      //
960030      for (zone = 0, m = 1; m < (SB_MAP_ZONE_SIZE * 16); m++)
960031        {
960032          map_element = m / 16;
960033          map_bit     = m % 16;
960034          map_mask    = 1 << map_bit;
960035          if (!(sb->map_zone[map_element] & map_mask))
960036            {
960037              //
960038              // Found a free place: set the map.
960039              //
960040              sb->map_zone[map_element] |= map_mask;
960041              sb->changed = 1;
960042              //
960043              // The *second* bit inside the map is for the first data
960044              // zone (the zone after the inode table inside the file
960045              // system), because the first is for a special 'zero' data
960046              // zone, not really used.
960047              //
960048              zone = sb->first_data_zone + m - 1; // Found a free zone.
960049              //
960050              // If the zone is outside the disk size, let set the map
960051              // bit, but reset variable 'zone'.
960052              //
960053              if (zone >= sb->zones)
960054                {
960055                  zone = 0;
960056                }
960057              else
960058                {
960059                  break;
960060                }
960061            }
960062        }
960063      if (zone == 0)
960064        {
960065          errset (ENOSPC);        // No space left on device.
960066          return ((zno_t) 0);
960067        }
960068      //
960069      // A free zone was found and the map was modified inside
960070      // the super block in memory. The zone must be cleared.
960071      //
960072      status = zone_write (sb, zone, buffer);
960073      if (status != 0)
960074        {
960075          zone_free (sb, zone);
960076          return ((zno_t) 0);
960077        }
960078      //
960079      // A zone was allocated: return the number.
960080      //
960081      return (zone);
960082  }
```

kernel/fs/zone_free.c

«

Si veda la sezione i159.3.51.

```
970001  #include <kernel/fs.h>
970002  #include <kernel/devices.h>
970003  #include <errno.h>
970004  //-------------------------------------------------------------------------
970005  int
970006  zone_free (sb_t *sb, zno_t zone)
970007  {
970008      int map_element;
970009      int map_bit;
970010      int map_mask;
970011      //
970012      // Check arguments.
970013      //
970014      if (sb == NULL || zone < sb->first_data_zone)
970015        {
970016          errset (EINVAL);        // Invalid argument.
970017          return (-1);
970018        }
970019      //
970020      // Calculate the map element, the map bit and the map mask.
970021      //
970022      // The *second* bit inside the map is for the first data-zone
970023      // (the zone after the inode table inside the file system),
970024      // because the first is for a special 'zero' data-zone, not
970025      // really used.
970026      //
970027      map_element = (zone - sb->first_data_zone + 1) / 16;
970028      map_bit     = (zone - sb->first_data_zone + 1) % 16;
970029      map_mask    = 1 << map_bit;
970030      //
970031      // Verify if the requested zone is inside the file system area.
970032      //
970033      if (zone >= sb->zones)
970034        {
970035          errset (EINVAL);        // Invalid argument.
970036          return (-1);
```

```
970037        }
970038      //
970039      // Free the zone and return.
970040      //
970041      if (sb->map_zone[map_element] & map_mask)
970042        {
970043          sb->map_zone[map_element] &= ~map_mask;
970044          sb->changed = 1;
970045          return (0);
970046        }
970047      else
970048        {
970049          errset (EUNKNOWN);              // The zone was already free.
970050          return (-1);
970051        }
970052  }
```

kernel/fs/zone_read.c

«

Si veda la sezione i159.3.53.

```
980001  #include <sys/os16.h>
980002  #include <kernel/fs.h>
980003  #include <kernel/devices.h>
980004  #include <errno.h>
980005  //------------------------------------------------------------------------
980006  int
980007  zone_read (sb_t *sb, zno_t zone, void *buffer)
980008  {
980009      size_t   size_zone;
980010      off_t    off_start;
980011      ssize_t  size_read;
980012      //
980013      // Verify if the requested zone is inside the file system area.
980014      //
980015      if (zone >= sb->zones)
980016        {
980017          errset (EINVAL);        // Invalid argument.
980018          return (-1);
980019        }
980020      //
980021      // Calculate start position.
980022      //
980023      size_zone  = 1024 << sb->log2_size_zone;
980024      off_start  = zone;
980025      off_start *= size_zone;
980026      //
980027      // Read from device to the buffer.
980028      //
980029      size_read = dev_io ((pid_t) -1, sb->device, DEV_READ, off_start,
980030                          buffer, size_zone, NULL);
980031      if (size_read != size_zone)
980032        {
980033          errset (EIO);           // I/O error.
980034          return (-1);
980035        }
980036      else
980037        {
980038          return (0);
980039        }
980040  }
```

kernel/fs/zone_write.c

«

Si veda la sezione i159.3.53.

```
990001  #include <kernel/fs.h>
990002  #include <kernel/devices.h>
990003  #include <errno.h>
990004  //------------------------------------------------------------------------
990005  int
990006  zone_write (sb_t *sb, zno_t zone, void *buffer)
990007  {
990008      size_t   size_zone;
990009      off_t    off_start;
990010      ssize_t  size_written;
990011      //
990012      // Verify if write is allowed.
990013      //
990014      if (sb->options & MOUNT_RO)
990015        {
990016          errset (EROFS);         // Read-only file system.
990017          return (-1);
990018        }
990019      //
990020      // Verify if the requested zone is inside the file system area.
990021      //
990022      if (zone >= sb->zones)
990023        {
990024          errset (EINVAL);        // Invalid argument.
990025          return (-1);
990026        }
990027      //
990028      // Write is allowed: calculate start position.
990029      //
990030      size_zone  = 1024 << sb->log2_size_zone;
990031      off_start  = zone;
990032      off_start *= size_zone;
990033      //
990034      // Write the buffer to the device.
```



```
990035      //
990036      size_written = dev_io ((pid_t) -1, sb->device, DEV_WRITE, off_start,
990037                             buffer, size_zone, NULL);
990038      if (size_written != size_zone)
990039        {
990040          errset (EIO);           // I/O error.
990041          return (-1);
990042        }
990043      else
990044        {
990045          return (0);
990046        }
990047  }
```

os16: «kernel/ibm_i86.h»

«

Si veda la sezione u0.4.

```
1000001  #ifndef _KERNEL_IBM_I86_H
1000002  #define _KERNEL_IBM_I86_H 1
1000003
1000004  #include <stdint.h>
1000005  #include <size_t.h>
1000006  #include <kernel/memory.h>
1000007  #include <sys/types.h>
1000008  //------------------------------------------------------------------------
1000009  #define IBM_I86_VIDEO_MODE   0x02
1000010  #define IBM_I86_VIDEO_PAGES  4
1000011
1000012  #define IBM_I86_VIDEO_COLUMNS   80
1000013  #define IBM_I86_VIDEO_ROWS      25
1000014  #define IBM_I86_VIDEO_ADDRESS   0xB8000L, 0xB9000L, 0xBA000L, 0xBB000L
1000015  //------------------------------------------------------------------------
1000016  void     _int10_00 (uint16_t video_mode);
1000017  void     _int10_02 (uint16_t page, uint16_t position);
1000018  void     _int10_05 (uint16_t page);
1000019  uint16_t _int12    (void);
1000020  uint16_t _int13_00 (uint16_t drive);
1000021  uint16_t _int13_02 (uint16_t drive, uint16_t sectors,
1000022                      uint16_t cylinder, uint16_t head,
1000023                      uint16_t sector, void *buffer);
1000024  uint16_t _int13_03 (uint16_t drive, uint16_t sectors,
1000025                      uint16_t cylinder, uint16_t head,
1000026                      uint16_t sector, void *buffer);
1000027  uint16_t _int16_00 (void);
1000028  uint16_t _int16_01 (void);
1000029  uint16_t _int16_02 (void);
1000030
1000031  #define int10_00(video_mode)     (_int10_00 ((uint16_t) video_mode))
1000032  #define int10_02(page, position) (_int10_02 ((uint16_t) page, \
1000033                                               (uint16_t) position))
1000034  #define int10_05(page)           (_int10_05 ((uint16_t) page))
1000035  #define int12()                  ((unsigned int) _int12 ())
1000036
1000037  #define int13_00(drive)          ((unsigned int) \
1000038                                    _int13_00 ((uint16_t) drive))
1000039  #define int13_02(drive, sectors, cylinder, head, sector, buffer) \
1000040                                   ((unsigned int) \
1000041                                    _int13_02 ((uint16_t) drive, \
1000042                                               (uint16_t) sectors, \
1000043                                               (uint16_t) cylinder, \
1000044                                               (uint16_t) head, \
1000045                                               (uint16_t) sector, \
1000046                                               buffer))
1000047  #define int13_03(drive, sectors, cylinder, head, sector, buffer) \
1000048                                   ((unsigned int) \
1000049                                    _int13_03 ((uint16_t) drive, \
1000050                                               (uint16_t) sectors, \
1000051                                               (uint16_t) cylinder, \
1000052                                               (uint16_t) head, \
1000053                                               (uint16_t) sector, \
1000054                                               buffer))
1000055  #define int16_00()               ((unsigned int) _int16_00 ())
1000056  #define int16_01()               ((unsigned int) _int16_01 ())
1000057  #define int16_02()               ((unsigned int) _int16_02 ())
1000058  //------------------------------------------------------------------------
1000059  uint16_t _in_8   (uint16_t port);
1000060  uint16_t _in_16  (uint16_t port);
1000061  void     _out_8  (uint16_t port, uint16_t value);
1000062  void     _out_16 (uint16_t port, uint16_t value);
1000063
1000064  #define in_8(port)           ((unsigned int) _in_8 ((uint16_t) port))
1000065  #define in_16(port)          ((unsigned int) _in_16 ((uint16_t) port))
1000066  #define out_8(port, value)   (_out_8 ((uint16_t) port, \
1000067                                        (uint16_t) value))
1000068  #define out_16(port, value)  (_out_16 ((uint16_t) port, \
1000069                                         (uint16_t) value))
1000070  //------------------------------------------------------------------------
1000071  void _cli     (void);
1000072  void _sti     (void);
1000073
1000074  #define cli() (_cli ())
1000075  #define sti() (_sti ())
1000076  //------------------------------------------------------------------------
1000077  void irq_on  (unsigned int irq);
1000078  void irq_off (unsigned int irq);
1000079  //------------------------------------------------------------------------
1000080  void     _ram_copy      (segment_t org_seg, offset_t org_off,
1000081                           segment_t dst_seg, offset_t dst_off,
1000082                           uint16_t size);
1000083
```

```
#define ram_copy(org_seg, org_off, dst_seg, dst_off, size) \
                        (_ram_copy ((uint16_t) org_seg, \
                                    (uint16_t) org_off, \
                                    (uint16_t) dst_seg, \
                                    (uint16_t) dst_off, \
                                    (uint16_t) size))
//---------------------------------------------------------------------
void con_select     (int console);
void con_putc       (int console, int c);
void con_scroll     (int console);
int  con_char_wait  (void);
int  con_char_read  (void);
int  con_char_ready (void);
void con_init       (void);
//---------------------------------------------------------------------
#define DSK_MAX                 4
#define DSK_SECTOR_SIZE         512     // Fixed!

typedef struct {
    unsigned int  bios_drive;
    unsigned int  cylinders;
    unsigned int  heads;
    unsigned int  sectors;
    unsigned int  retry;
} dsk_t;

typedef struct {
    unsigned int  cylinder;
    unsigned int  head;
    unsigned int  sector;
} dsk_chs_t;
//---------------------------------------------------------------------
extern dsk_t dsk_table[DSK_MAX];
//---------------------------------------------------------------------
void   dsk_setup         (void);
int    dsk_reset         (int drive);
void   dsk_sector_to_chs (int drive, unsigned int sector,
                          dsk_chs_t *chs);
int    dsk_read_sectors  (int drive, unsigned int start_sector,
                          void *buffer, unsigned int n_sectors);
int    dsk_write_sectors (int drive, unsigned int start_sector,
                          void *buffer, unsigned int n_sectors);
size_t dsk_read_bytes    (int drive, off_t offset,
                          void *buffer, size_t count);
size_t dsk_write_bytes   (int drive, off_t offset,
                          void *buffer, size_t count);
//---------------------------------------------------------------------

#endif
```

kernel/ibm_i86/_cli.s

«

Si veda la sezione u0.4.

```
.global __cli
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; Clear interrupt flag.
;---------------------------------------------------------------------
.align 2
__cli:
    cli
    ret
```

kernel/ibm_i86/_in_16.s

«

Si veda la sezione u0.4.

```
.global __in_16
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; Port input word.
;---------------------------------------------------------------------
__in_16:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   dx, 4[bp]     ; 1st arg (port number).
    in    ax, dx
    mov   -2[bp], ax    ; Save AX.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```

kernel/ibm_i86/_in_8.s

«

Si veda la sezione u0.4.

```
.global __in_8
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; Port input byte.
;---------------------------------------------------------------------
__in_8:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   dx, 4[bp]     ; 1st arg (port number).
    in    al, dx
    mov   ah, #0
    mov   -2[bp], ax    ; Save AX.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```

kernel/ibm_i86/_int10_00.s

«

Si veda la sezione u0.4.

```
.global __int10_00
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; INT 0x10 - video - set video mode
;       AH = 0x00
;       AL = desired video mode:
;            0x00 = text 40x25  pages 8
;            0x01 = text 40x25  pages 8
;            0x02 = text 80x25  pages 4
;            0x03 = text 80x25  pages 4
;            0x07 = text 80x25  pages 4?
;
; Specify the display mode for the currently active display adapter.
;---------------------------------------------------------------------
.align 2
__int10_00:
    enter #0, #0        ; No local variables.
    pushf
    cli
    pusha
    mov   ah, #0x00
    mov   al, 4[bp]     ; 1st arg (video mode).
    int   #0x10
    popa
    popf
    leave
    ret
```

kernel/ibm_i86/_int10_02.s

«

Si veda la sezione u0.4.

```
.global __int10_02
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; INT 0x10 - video - set cursor position
;       AH = 0x02
;       BH = page number:
;            0-7 in modes 0 and 1
;            0-3 in modes 2 and 3
;       DH = row (0x00 is top)
;       DL = column (0x00 is left)
;---------------------------------------------------------------------
.align 2
__int10_02:
    enter #0, #0        ; No local variables.
    pushf
    cli
    pusha
    mov   ah, #0x02
    mov   bh, #0x00
    mov   bh, 4[bp]     ; 1st arg (page).
    mov   dx, 6[bp]     ; 2nd arg (pos).
    int   #0x10
    popa
    popf
    leave
    ret
```

kernel/ibm_i86/_int10_05.s

«

Si veda la sezione u0.4.

```
.global __int10_05
;---------------------------------------------------------------------
.text
;---------------------------------------------------------------------
; INT 0x10 - video - select active display page
;       AH = 0x05
;       AL = new page number (0x00 is the first)
;---------------------------------------------------------------------
.align 2
__int10_05:
    enter #0, #0        ; No local variables.
    pushf
    cli
```

```
1060014     pusha
1060015     mov   ah, #0x05
1060016     mov   bh, 4[bp]     ; 1st arg (page).
1060017     int   #0x10
1060018     popa
1060019     popf
1060020     leave
1060021     ret
```

kernel/ibm_i86/_int12.s

«

Si veda la sezione u0.4.

```
1070001 .global __int12
1070002 ;--------------------------------------------------------------------
1070003 .text
1070004 ;--------------------------------------------------------------------
1070005 ; INT 12 - bios - get memory size
1070006 ; Return:
1070007 ;       AX = kilobytes of contiguous memory starting at absolute address
1070008 ;            0x00000
1070009 ;
1070010 ; This call returns the contents of the word at absolute address
1070011 ; 0x00413.
1070012 ;--------------------------------------------------------------------
1070013 .align 2
1070014 __int12:
1070015     enter #2, #0        ; 1 local variable.
1070016     pushf
1070017     cli
1070018     pusha
1070019     int   #0x12
1070020     mov   -2[bp], ax    ; save AX.
1070021     popa
1070022     popf
1070023     mov   ax, -2[bp]    ; AX is the function return value.
1070024     leave
1070025     ret
```

kernel/ibm_i86/_int13_00.s

«

Si veda la sezione u0.4.

```
1080001 .global __int13_00
1080002 ;--------------------------------------------------------------------
1080003 .text
1080004 ;--------------------------------------------------------------------
1080005 ; INT 0x13 - disk - reset disk system
1080006 ;       AH = 0x00
1080007 ;       DL = drive (if bit 7 is set both hard disks and floppy disks
1080008 ;            reset)
1080009 ; Return:
1080010 ;       AH = status
1080011 ;       CF clear if successful (returned AH=0x00)
1080012 ;       CF set on error
1080013 ;--------------------------------------------------------------------
1080014 .align 2
1080015 __int13_00:
1080016     enter #2, #0        ; 1 local variable.
1080017     pushf
1080018     cli
1080019     pusha
1080020     mov   ah, #0x00
1080021     mov   dl, 4[bp]     ; 1st arg.
1080022     int   #0x13
1080023     mov   al, #0x00
1080024     mov   -2[bp], ax    ; save AX.
1080025     popa
1080026     popf
1080027     mov   ax, -2[bp]    ; AX is the function return value.
1080028     leave
1080029     ret
```

kernel/ibm_i86/_int13_02.s

«

Si veda la sezione u0.4.

```
1090001 .global __int13_02
1090002 ;--------------------------------------------------------------------
1090003 .text
1090004 ;--------------------------------------------------------------------
1090005 ; INT 0x13 - disk - read sectors into memory
1090006 ;       AH = 0x02
1090007 ;       AL = number of sectors to read (must be nonzero)
1090008 ;       CH = cylinder number (0-255)
1090009 ;       CL bit 6-7 =
1090010 ;            cylinder number (256-1023)
1090011 ;       CL bit 0-5 =
1090012 ;            sector number   (1-63)
1090013 ;       DH = head number     (0-255)
1090014 ;       DL = drive number (bit 7 set for hard disk)
1090015 ;       ES:BX -> data buffer
1090016 ; Return:
1090017 ;       CF set on error
1090018 ;       CF clear if successful
1090019 ;       AH = status (0x00 if successful)
1090020 ;       AL = number of sectors transferred (only valid if CF set for
1090021 ;            some BIOSes)
```



```
1090022 ;--------------------------------------------------------------------
1090023 .align 2
1090024 __int13_02:
1090025     enter #2, #0        ; 1 local variable.
1090026     pushf
1090027     cli
1090028     pusha
1090029     mov   ax, ds        ; Set ES the same as DS.
1090030     mov   es, ax        ;
1090031     mov   ax,  8[bp]    ; 3rd arg (cylinder). It must be splitted and
1090032     mov   ch, al        ; assigned to the right registers.
1090033     mov   cl, ah        ;
1090034     shl   cl, 1         ;
1090035     shl   cl, 1         ;
1090036     shl   cl, 1         ;
1090037     shl   cl, 1         ;
1090038     shl   cl, 1         ;
1090039     shl   cl, 1         ;
1090040     add   cl, 12[bp]    ; 5th arg (sector).
1090041     mov   dl,  4[bp]    ; 1st arg (drive).
1090042     mov   al,  6[bp]    ; 2nd arg (sectors to be read).
1090043     mov   dh, 10[bp]    ; 4th arg (head).
1090044     mov   bx, 14[bp]    ; 6th arg (buffer pointer).
1090045     mov   ah, #0x02
1090046     int   #0x13
1090047     mov   -2[bp], ax    ; save AX.
1090048     popa
1090049     popf
1090050     mov   ax, -2[bp]    ; AX is the function return value.
1090051     leave
1090052     ret
```

kernel/ibm_i86/_int13_03.s

«

Si veda la sezione u0.4.

```
1100001 .global __int13_03
1100002 ;--------------------------------------------------------------------
1100003 .text
1100004 ;--------------------------------------------------------------------
1100005 ; INT 0x13 - disk - write sectors to disk
1100006 ;       AH = 0x03
1100007 ;       AL = number of sectors to write (must be nonzero)
1100008 ;       CH = cylinder number (0-255)
1100009 ;       CL bit 6-7 =
1100010 ;            cylinder number (256-1023)
1100011 ;       CL bit 0-5 =
1100012 ;            sector number   (1-63)
1100013 ;       DH = head number     (0-255)
1100014 ;       DL = drive number (bit 7 set for hard disk)
1100015 ;       ES:BX -> data buffer
1100016 ; Return:
1100017 ;       CF set on error
1100018 ;       CF clear if successful
1100019 ;       AH = status (0x00 if successful)
1100020 ;       AL = number of sectors transferred (only valid if CF set for
1100021 ;            some BIOSes)
1100022 ;--------------------------------------------------------------------
1100023 .align 2
1100024 __int13_03:
1100025     enter #2, #0        ; 1 local variable.
1100026     pushf
1100027     cli
1100028     pusha
1100029     mov   ax, ds        ; Set ES the same as DS.
1100030     mov   es, ax        ;
1100031     mov   ax,  8[bp]    ; 3rd arg (cylinder). It must be splitted and
1100032     mov   ch, al        ; assigned to the right registers.
1100033     mov   cl, ah        ;
1100034     shl   cl, 1         ;
1100035     shl   cl, 1         ;
1100036     shl   cl, 1         ;
1100037     shl   cl, 1         ;
1100038     shl   cl, 1         ;
1100039     shl   cl, 1         ;
1100040     add   cl, 12[bp]    ; 5th arg (sector).
1100041     mov   dl,  4[bp]    ; 1st arg (drive).
1100042     mov   al,  6[bp]    ; 2nd arg (sectors to be written).
1100043     mov   dh, 10[bp]    ; 4th arg (head).
1100044     mov   bx, 14[bp]    ; 6th arg (buffer pointer).
1100045     mov   ah, #0x03
1100046     int   #0x13
1100047     mov   -2[bp], ax    ; save AX.
1100048     popa
1100049     popf
1100050     mov   ax, -2[bp]    ; AX is the function return value.
1100051     leave
1100052     ret
```

kernel/ibm_i86/_int16_00.s

«

Si veda la sezione u0.4.

```
1110001 .global __int16_00
1110002 ;--------------------------------------------------------------------
1110003 .text
1110004 ;--------------------------------------------------------------------
1110005 ; INT 0x16 - keyboard - get keystroke
1110006 ;       AH = 0x00
1110007 ; Return:
```

```
;       AH = BIOS scan code
;       AL = ASCII character
;--------------------------------------------------------------------------
.align 2
__int16_00:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   ah, #0x00
    int   #0x16
    mov   -2[bp], ax    ; Save AX.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```

kernel/ibm_i86/_int16_01.s

«

Si veda la sezione u0.4.

```
.global __int16_01
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; INT 0x16 - keyboard - check for keystroke
;       AH = 0x01
; Return:
;       ZF set if no keystroke available
;       ZF clear if keystroke available
;           AH = BIOS scan code
;           AL = ASCII character
;
; If a keystroke is present, it is not removed from the keyboard buffer.
;--------------------------------------------------------------------------
.align 2
__int16_01:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   ah, #0x01
    int   #0x16
    jnz   __int16_01_ok
    mov   ax, #0        ; Put zero to AX, if no keystroke is available.
__int16_01_ok:
    mov   -2[bp], ax    ; Save AX.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```

kernel/ibm_i86/_int16_02.s

«

Si veda la sezione u0.4.

```
.global __int16_02
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; INT 0x16 - keyboard - get shift flags
;       AH = 0x02
; Return:
;       AL = shift flags
;       AH might be destroyed
;
; bit 7    Insert active
; bit 6    CapsLock active
; bit 5    NumLock active
; bit 4    ScrollLock active
; bit 3    Alt key pressed
; bit 2    Ctrl key pressed
; bit 1    left shift key pressed
; bit 0    right shift key pressed
;--------------------------------------------------------------------------
.align 2
__int16_02:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   ah, #0x02
    int   #0x16
    mov   ah, #0        ; Reset AH.
    mov   -2[bp], ax    ; Save AX.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```


kernel/ibm_i86/_out_16.s

«

Si veda la sezione u0.4.

```
.global __out_16
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; Port output word.
;--------------------------------------------------------------------------
.align 2
__out_16:
    enter #0, #0        ; No local variables.
    pushf
    cli
    pusha
    mov   dx, 4[bp]     ; 1st arg (port number).
    mov   ax, 6[bp]     ; 2nd arg (value).
    out   dx, ax
    popa
    popf
    leave
    ret
```

kernel/ibm_i86/_out_8.s

«

Si veda la sezione u0.4.

```
.global __out_8
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; Port output byte.
;--------------------------------------------------------------------------
.align 2
__out_8:
    enter #0, #0        ; No local variables.
    pushf
    cli
    pusha
    mov   dx, 4[bp]     ; 1st arg (port number).
    mov   ax, 6[bp]     ; 2nd arg (value).
    out   dx, al
    popa
    popf
    leave
    ret
```

kernel/ibm_i86/_ram_copy.s

«

Si veda la sezione u0.4.

```
.global __ram_copy
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; Copy some bytes between segments.
;--------------------------------------------------------------------------
.align 2
__ram_copy:
    enter #0, #0        ; No local variables.
    pushf
    cli
    pusha
    mov   ax,  4[bp]    ; 1st arg (source segment).
    mov   si,  6[bp]    ; 2nd arg (source offset).
    mov   bx,  8[bp]    ; 3rd arg (destination segment).
    mov   di, 10[bp]    ; 4th arg (destination offset).
    mov   cx, 12[bp]    ; 5th arg (size).
    mov   dx, ds        ; save the data segment.
    mov   ds, ax        ; set DS.
    mov   es, bx        ; set ES.
    rep
    movsb               ; Copy the array of bytes.
    mov   ds, dx        ; Restore the data segment.
    mov   es, dx        ; Restore or fix the extra segment.
    popa
    popf
    leave
    ret
```

kernel/ibm_i86/_sti.s

«

Si veda la sezione u0.4.

```
.global __sti
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; Set interrupt flag.
;--------------------------------------------------------------------------
.align 2
__sti:
    sti
    ret
```

kernel/ibm_i86/con_char_read.c

«

Si veda la sezione u0.4.

```
1180001  #include <kernel/ibm_i86.h>
1180002  #include <kernel/k_libc.h>
1180003  #include <sys/os16.h>
1180004  #include <sys/types.h>
1180005  #include <stdint.h>
1180006  //-------------------------------------------------------------------------
1180007  int
1180008  con_char_read (void)
1180009  {
1180010      int c;
1180011      c = int16_01 ();
1180012      //
1180013      // Remove special keys that are not used: they have zero in the low
1180014      // 8 bits, and something in the upper 8 bits.
1180015      //
1180016      if ((c & 0xFF00) && !(c & 0x00FF))
1180017        {
1180018          int16_00 ();    // Remove from buffer and return zero:
1180019          return (0);     // no key.
1180020        }
1180021      //
1180022      // A common key was pressed: filter only che low 8 bits.
1180023      //
1180024      c = c & 0x00FF;
1180025      if (c == 0)
1180026        {
1180027          return (c);     // There is no key.
1180028        }
1180029      if (c == '\r')      // Convert 'CR' to 'LF'.
1180030        {
1180031          c = '\n';
1180032        }
1180033      int16_00 ();        // Remove the key from buffer and return.
1180034      return (c);
1180035  }
```

kernel/ibm_i86/con_char_ready.c

«

Si veda la sezione u0.4.

```
1190001  #include <kernel/ibm_i86.h>
1190002  #include <kernel/k_libc.h>
1190003  #include <sys/os16.h>
1190004  #include <sys/types.h>
1190005  #include <stdint.h>
1190006  //-------------------------------------------------------------------------
1190007  int
1190008  con_char_ready (void)
1190009  {
1190010      int c;
1190011      c = int16_01 ();
1190012      //
1190013      // Remove special keys that are not used: they have zero in the low
1190014      // 8 bits, and something in the upper 8 bits.
1190015      //
1190016      if ((c & 0xFF00) && !(c & 0x00FF))
1190017        {
1190018          int16_00 ();    // Remove from buffer and return zero:
1190019          return (0);     // no key.
1190020        }
1190021      //
1190022      // A common key was pressed: filter only che low 8 bits.
1190023      //
1190024      c = c & 0x00FF;
1190025      return (c);
1190026  }
```

kernel/ibm_i86/con_char_wait.c

«

Si veda la sezione u0.4.

```
1200001  #include <kernel/ibm_i86.h>
1200002  #include <kernel/k_libc.h>
1200003  #include <sys/os16.h>
1200004  #include <sys/types.h>
1200005  #include <stdint.h>
1200006  //-------------------------------------------------------------------------
1200007  int
1200008  con_char_wait (void)
1200009  {
1200010      int c;
1200011      c = int16_00 ();
1200012      c = c & 0x00FF;
1200013      if (c == '\r')
1200014        {
1200015          c = '\n';
1200016        }
1200017      return (c);
1200018  }
```


kernel/ibm_i86/con_init.c

«

Si veda la sezione u0.4.

```
1210001  #include <kernel/ibm_i86.h>
1210002  #include <kernel/k_libc.h>
1210003  #include <sys/os16.h>
1210004  #include <sys/types.h>
1210005  #include <stdint.h>
1210006  //-------------------------------------------------------------------------
1210007  void
1210008  con_init (void)
1210009  {
1210010      int page;
1210011      //
1210012      int10_00 (IBM_I86_VIDEO_MODE);
1210013      int10_05 (0);
1210014      //
1210015      for (page = 0; page < IBM_I86_VIDEO_PAGES; page++)
1210016        {
1210017          con_putc (page, '\n');
1210018        }
1210019  }
```

kernel/ibm_i86/con_putc.c

«

Si veda la sezione u0.4.

```
1220001  #include <kernel/ibm_i86.h>
1220002  #include <kernel/k_libc.h>
1220003  #include <sys/os16.h>
1220004  #include <sys/types.h>
1220005  #include <stdint.h>
1220006  //-------------------------------------------------------------------------
1220007  void
1220008  con_putc (int console, int c)
1220009  {
1220010      static int      cursor[IBM_I86_VIDEO_PAGES];
1220011      static addr_t   address[] = {IBM_I86_VIDEO_ADDRESS};
1220012             addr_t   address_destination;
1220013             size_t   size_screen;
1220014             size_t   size_row;
1220015             uint16_t cell;
1220016             uint16_t attribute = 0x0700;
1220017             int      cursor_row;
1220018             int      cursor_column;
1220019             int      cursor_combined;
1220020
1220021      if (console < 0 || console >= IBM_I86_VIDEO_PAGES)
1220022        {
1220023          //
1220024          // No such console.
1220025          //
1220026          return;
1220027        }
1220028      //
1220029      // Calculate sizes.
1220030      //
1220031      size_row    = IBM_I86_VIDEO_COLUMNS;
1220032      size_screen = size_row * IBM_I86_VIDEO_ROWS;
1220033      //
1220034      // See if it is a special character, or if the cursor position
1220035      // requires a scroll up.
1220036      //
1220037      if (c == '\n')
1220038        {
1220039          con_scroll (console);
1220040          cursor[console] = (size_screen - size_row);
1220041        }
1220042      else if (c == '\b')
1220043        {
1220044          cursor[console]--;
1220045          if (cursor[console] < 0)
1220046            {
1220047              cursor[console] = 0;
1220048            }
1220049        }
1220050      else if (cursor[console] == (size_screen -1))
1220051        {
1220052          //
1220053          // Scroll up.
1220054          //
1220055          con_scroll (console);
1220056          //
1220057          cursor[console] -= size_row;
1220058        }
1220059      //
1220060      // If it is not a control character, print it.
1220061      //
1220062      if (c != '\n' && c != '\b')
1220063        {
1220064          //
1220065          // Write the character.
1220066          //
1220067          address_destination  = address[console];
1220068          address_destination += (cursor[console] * 2);
1220069          cell = (attribute | (c & 0x00FF));
1220070          //
1220071          mem_write (address_destination, &cell, sizeof (uint16_t));
1220072          //
1220073          // and an extra space after it (to be able to show the cursor).
```

```
1220074         //
1220075         cell = (attribute | ' ');
1220076         address_destination  += 2;
1220077         mem_write (address_destination, &cell, sizeof (uint16_t));
1220078         //
1220079         //
1220080         //
1220081         cursor[console]++;
1220082       }
1220083     //
1220084     // Update the cursor position on screen.
1220085     //
1220086     cursor_row      = cursor[console] / size_row;
1220087     cursor_column   = cursor[console] % size_row;
1220088     cursor_combined = (cursor_row << 8) | cursor_column;
1220089     //
1220090     // Set cursor position.
1220091     //
1220092     int10_02 (console, cursor_combined);
1220093 }
```

kernel/ibm_i86/con_scroll.c

«

Si veda la sezione u0.4.

```
1230001 #include <kernel/ibm_i86.h>
1230002 #include <kernel/k_libc.h>
1230003 #include <sys/os16.h>
1230004 #include <sys/types.h>
1230005 #include <stdint.h>
1230006 //---------------------------------------------------------------------------
1230007 void
1230008 con_scroll (int console)
1230009 {
1230010     static addr_t   address[] = {IBM_I86_VIDEO_ADDRESS};
1230011            addr_t   address_source;
1230012            addr_t   address_destination;
1230013            size_t   size_screen;
1230014            size_t   size_row;
1230015     static uint16_t empty_line[IBM_I86_VIDEO_COLUMNS];
1230016     //
1230017     size_row    = IBM_I86_VIDEO_COLUMNS;
1230018     size_screen = size_row * IBM_I86_VIDEO_ROWS;
1230019     //
1230020     // Scroll up.
1230021     //
1230022     address_source      = address[console];
1230023     address_source     += size_row * 2;
1230024     address_destination = address[console];
1230025     //
1230026     mem_copy (address_source, address_destination,
1230027               (size_t) ((size_screen - size_row) * 2));
1230028     //
1230029     address_destination  = address[console];
1230030     address_destination += ((size_screen - size_row) * 2);
1230031     //
1230032     mem_write (address_destination, &empty_line,
1230033                (size_t) (size_row * 2));
1230034 }
```

kernel/ibm_i86/con_select.c

«

Si veda la sezione u0.4.

```
1240001 #include <kernel/ibm_i86.h>
1240002 #include <kernel/k_libc.h>
1240003 #include <sys/os16.h>
1240004 #include <sys/types.h>
1240005 #include <stdint.h>
1240006 //---------------------------------------------------------------------------
1240007 void
1240008 con_select (int console)
1240009 {
1240010     //
1240011     // Variable 'console' gose from zero to 'IBM_I86_VIDEO_PAGES - 1'.
1240012     //
1240013     if (console >= 0 && console < IBM_I86_VIDEO_PAGES)
1240014       {
1240015         int10_05 (console);
1240016       }
1240017 }
```

kernel/ibm_i86/dsk_read_bytes.c

«

Si veda la sezione u0.4.

```
1250001 #include <kernel/ibm_i86.h>
1250002 #include <kernel/k_libc.h>
1250003 #include <sys/os16.h>
1250004 #include <sys/types.h>
1250005 #include <stdint.h>
1250006 //---------------------------------------------------------------------------
1250007 size_t
1250008 dsk_read_bytes (int drive, off_t offset, void *buffer, size_t count)
1250009 {
1250010     unsigned char *data_buffer = (unsigned char *) buffer;
1250011     int            status;
1250012     unsigned int   sector;
```



```
1250013     unsigned char  sector_buffer[DSK_SECTOR_SIZE];
1250014     int            i;
1250015     int            j = 0;
1250016     size_t         k = 0;
1250017
1250018     sector = offset / DSK_SECTOR_SIZE;
1250019     i      = offset % DSK_SECTOR_SIZE;
1250020
1250021     status = dsk_read_sectors (drive, sector, sector_buffer, 1);
1250022
1250023     if (status != 0)
1250024       {
1250025         return ((size_t) 0);
1250026       }
1250027
1250028     while (count)
1250029       {
1250030         for (; i < DSK_SECTOR_SIZE && count > 0;
1250031                i++, j++, k++, count--, offset++)
1250032           {
1250033             data_buffer[j] = sector_buffer[i];
1250034           }
1250035         if (count)
1250036           {
1250037             sector = offset / DSK_SECTOR_SIZE;
1250038             i      = offset % DSK_SECTOR_SIZE;
1250039             status = dsk_read_sectors (drive, sector, sector_buffer, 1);
1250040             if (status != 0)
1250041               {
1250042                 return (k);
1250043               }
1250044           }
1250045       }
1250046     return (k);
1250047 }
```

kernel/ibm_i86/dsk_read_sectors.c

«

Si veda la sezione u0.4.

```
1260001 #include <kernel/ibm_i86.h>
1260002 #include <kernel/k_libc.h>
1260003 #include <sys/os16.h>
1260004 #include <sys/types.h>
1260005 #include <stdint.h>
1260006 //---------------------------------------------------------------------------
1260007 int
1260008 dsk_read_sectors (int drive, unsigned int start_sector, void *buffer,
1260009                   unsigned int n_sectors)
1260010 {
1260011     int          status;
1260012     unsigned int retry;
1260013     unsigned int remaining;
1260014     dsk_chs_t    chs;
1260015     dsk_sector_to_chs (drive, start_sector, &chs);
1260016     remaining = dsk_table[drive].sectors - chs.sector + 1;
1260017     if (remaining < n_sectors)
1260018       {
1260019         status = dsk_read_sectors (drive, start_sector, buffer,
1260020                                    remaining);
1260021         if (status == 0)
1260022           {
1260023             status = dsk_read_sectors (drive, start_sector + remaining,
1260024                                        buffer, n_sectors - remaining);
1260025           }
1260026         return (status);
1260027       }
1260028     else
1260029       {
1260030         for (retry = 0; retry < dsk_table[drive].retry; retry++)
1260031           {
1260032             status = int13_02 (dsk_table[drive].bios_drive, n_sectors,
1260033                                chs.cylinder, chs.head, chs.sector,
1260034                                buffer);
1260035             status = status & 0x00F0;
1260036             if (status == 0)
1260037               {
1260038                 break;
1260039               }
1260040             else
1260041               {
1260042                 dsk_reset (drive);
1260043               }
1260044           }
1260045       }
1260046     if (status == 0)
1260047       {
1260048         return (0);
1260049       }
1260050     else
1260051       {
1260052         return (-1);
1260053       }
1260054 }
```

kernel/ibm_i86/dsk_reset.c «

Si veda la sezione u0.4.

```
1270001  #include <kernel/ibm_i86.h>
1270002  #include <kernel/k_libc.h>
1270003  #include <sys/os16.h>
1270004  #include <sys/types.h>
1270005  #include <stdint.h>
1270006  //-------------------------------------------------------------------------
1270007  int
1270008  dsk_reset (int drive)
1270009  {
1270010      unsigned int status;
1270011      status = int13_00 (dsk_table[drive].bios_drive);
1270012      if (status == 0)
1270013        {
1270014          return (0);
1270015        }
1270016      else
1270017        {
1270018          return (-1);
1270019        }
1270020  }
```

kernel/ibm_i86/dsk_sector_to_chs.c «

Si veda la sezione u0.4.

```
1280001  #include <kernel/ibm_i86.h>
1280002  #include <kernel/k_libc.h>
1280003  #include <sys/os16.h>
1280004  #include <sys/types.h>
1280005  #include <stdint.h>
1280006  //-------------------------------------------------------------------------
1280007  void
1280008  dsk_sector_to_chs (int drive, unsigned int sector, dsk_chs_t *chs)
1280009  {
1280010      unsigned int sectors_per_cylinder;
1280011      sectors_per_cylinder = (dsk_table[drive].sectors
1280012                              * dsk_table[drive].heads);
1280013      chs->cylinder        = sector / sectors_per_cylinder;
1280014      sector               = sector % sectors_per_cylinder;
1280015      chs->head            = sector / dsk_table[drive].sectors;
1280016      sector               = sector % dsk_table[drive].sectors;
1280017      chs->sector          = sector + 1;
1280018  }
```

kernel/ibm_i86/dsk_setup.c «

Si veda la sezione u0.4.

```
1290001  #include <kernel/ibm_i86.h>
1290002  //-------------------------------------------------------------------------
1290003  void
1290004  dsk_setup (void)
1290005  {
1290006    dsk_reset (0);
1290007    dsk_table[0].bios_drive = 0x00;  // A: 1440 Kibyte floppy disk.
1290008    dsk_table[0].cylinders  = 80;
1290009    dsk_table[0].heads      = 2;
1290010    dsk_table[0].sectors    = 18;
1290011    dsk_table[0].retry      = 3;
1290012    dsk_reset (1);
1290013    dsk_table[1].bios_drive = 0x01;  // B: 1440 Kibyte floppy disk.
1290014    dsk_table[1].cylinders  = 80;
1290015    dsk_table[1].heads      = 2;
1290016    dsk_table[1].sectors    = 18;
1290017    dsk_table[1].retry      = 3;
1290018    dsk_reset (2);
1290019    dsk_table[2].bios_drive = 0x80;  // C: like a 2880 Kibyte floppy disk.
1290020    dsk_table[2].cylinders  = 80;
1290021    dsk_table[2].heads      = 2;
1290022    dsk_table[2].sectors    = 36;
1290023    dsk_table[2].retry      = 3;
1290024    dsk_reset (3);
1290025    dsk_table[3].bios_drive = 0x81;  // D: like a 2880 Kibyte floppy disk.
1290026    dsk_table[3].cylinders  = 80;
1290027    dsk_table[3].heads      = 2;
1290028    dsk_table[3].sectors    = 36;
1290029    dsk_table[3].retry      = 3;
1290030  }
```

kernel/ibm_i86/dsk_table.c «

Si veda la sezione u0.4.

```
1300001  #include <kernel/ibm_i86.h>
1300002  //-------------------------------------------------------------------------
1300003  dsk_t dsk_table[DSK_MAX];
```

kernel/ibm_i86/dsk_write_bytes.c «

Si veda la sezione u0.4.

```
1310001  #include <kernel/ibm_i86.h>
1310002  #include <kernel/k_libc.h>
1310003  #include <sys/os16.h>
```



```
1310004  #include <sys/types.h>
1310005  #include <stdint.h>
1310006  //-------------------------------------------------------------------------
1310007  size_t
1310008  dsk_write_bytes (int drive, off_t offset, void *buffer, size_t count)
1310009  {
1310010      unsigned char *data_buffer = (unsigned char *) buffer;
1310011      int            status;
1310012      unsigned int   sector;
1310013      unsigned char  sector_buffer[DSK_SECTOR_SIZE];
1310014      int            i;
1310015      int            j = 0;
1310016      size_t         k = 0;
1310017      size_t         m = 0;
1310018
1310019      sector = offset / DSK_SECTOR_SIZE;
1310020      i      = offset % DSK_SECTOR_SIZE;
1310021      status = dsk_read_sectors (drive, sector, sector_buffer, 1);
1310022
1310023      if (status != 0)
1310024        {
1310025          return ((size_t) 0);
1310026        }
1310027
1310028      while (count)
1310029        {
1310030          m = k;
1310031          for (; i < DSK_SECTOR_SIZE && count > 0;
1310032                 i++, j++, k++, count--, offset++)
1310033            {
1310034              sector_buffer[i] = data_buffer[j];
1310035            }
1310036          status = dsk_write_sectors (drive, sector, sector_buffer, 1);
1310037          if (status != 0)
1310038            {
1310039              return (m);
1310040            }
1310041          if (count)
1310042            {
1310043              sector = offset / DSK_SECTOR_SIZE;
1310044              i      = offset % DSK_SECTOR_SIZE;
1310045              status = dsk_read_sectors (drive, sector, sector_buffer, 1);
1310046              if (status != 0)
1310047                {
1310048                  return (m);
1310049                }
1310050            }
1310051        }
1310052      return (k);
1310053  }
1310054
1310055
```

kernel/ibm_i86/dsk_write_sectors.c «

Si veda la sezione u0.4.

```
1320001  #include <kernel/ibm_i86.h>
1320002  #include <kernel/k_libc.h>
1320003  #include <sys/os16.h>
1320004  #include <sys/types.h>
1320005  #include <stdint.h>
1320006  //-------------------------------------------------------------------------
1320007  int
1320008  dsk_write_sectors (int drive, unsigned int start_sector, void *buffer,
1320009                     unsigned int n_sectors)
1320010  {
1320011      int          status;
1320012      unsigned int retry;
1320013      unsigned int remaining;
1320014      dsk_chs_t    chs;
1320015      dsk_sector_to_chs (drive, start_sector, &chs);
1320016      remaining = dsk_table[drive].sectors - chs.sector + 1;
1320017      if (remaining < n_sectors)
1320018        {
1320019          status = dsk_write_sectors (drive, start_sector,
1320020                                      buffer, remaining);
1320021          if (status == 0)
1320022            {
1320023              status = dsk_write_sectors (drive,
1320024                                          start_sector + remaining,
1320025                                          buffer, n_sectors - remaining);
1320026            }
1320027          return (status);
1320028        }
1320029      else
1320030        {
1320031          for (retry = 0; retry < dsk_table[drive].retry; retry++)
1320032            {
1320033              status = int13_03 (dsk_table[drive].bios_drive, n_sectors,
1320034                                 chs.cylinder, chs.head, chs.sector,
1320035                                 buffer);
1320036              status = status & 0x00F0;
1320037              if (status == 0)
1320038                {
1320039                  break;
1320040                }
1320041              else
1320042                {
1320043                  dsk_reset (drive);
1320044                }
```

```
1320045           }
1320046         }
1320047       if (status == 0)
1320048         {
1320049           return (0);
1320050         }
1320051       else
1320052         {
1320053           return (-1);
1320054         }
1320055 }
```

kernel/ibm_i86/irq_off.c

«

Si veda la sezione u0.4.

```
1330001 #include <kernel/ibm_i86.h>
1330002 #include <kernel/k_libc.h>
1330003 #include <sys/os16.h>
1330004 #include <sys/types.h>
1330005 #include <stdint.h>
1330006 //---------------------------------------------------------------------
1330007 void
1330008 irq_off (unsigned int irq)
1330009 {
1330010     unsigned int mask;
1330011     unsigned int status;
1330012     if (irq > 7)
1330013       {
1330014         return; // Only XT IRQs are handled.
1330015       }
1330016     else
1330017       {
1330018         mask   = (1 << irq);
1330019         status = in_8 (0x21);
1330020         status = status | mask;
1330021         out_8 (0x21, status);
1330022       }
1330023 }
```

kernel/ibm_i86/irq_on.c

«

Si veda la sezione u0.4.

```
1340001 #include <kernel/ibm_i86.h>
1340002 #include <kernel/k_libc.h>
1340003 #include <sys/os16.h>
1340004 #include <sys/types.h>
1340005 #include <stdint.h>
1340006 //---------------------------------------------------------------------
1340007 void
1340008 irq_on (unsigned int irq)
1340009 {
1340010     unsigned int mask;
1340011     unsigned int status;
1340012     if (irq > 7)
1340013       {
1340014         return; // Only XT IRQs are handled.
1340015       }
1340016     else
1340017       {
1340018         mask   = ~(1 << irq);
1340019         status = in_8 (0x21);
1340020         status = status & mask;
1340021         out_8 (0x21, status);
1340022       }
1340023 }
```

os16: «kernel/k_libc.h»

«

Si veda la sezione u0.5.

```
1350001 #ifndef _KERNEL_K_LIBC_H
1350002 #define _KERNEL_K_LIBC_H       1
1350003
1350004 #include <const.h>
1350005 #include <restrict.h>
1350006 #include <size_t.h>
1350007 #include <clock_t.h>
1350008 #include <time_t.h>
1350009 #include <sys/types.h>
1350010 #include <stdarg.h>
1350011
1350012 //---------------------------------------------------------------------
1350013 void    k_exit     (int status);
1350014 //---------------------------------------------------------------------
1350015 clock_t k_clock    (void);
1350016 int     k_stime    (time_t *timer);
1350017 time_t  k_time     (time_t *timer);
1350018 //---------------------------------------------------------------------
1350019 int     k_puts     (const char *string);
1350020 int     k_printf   (const char *restrict format, ...);
1350021 int     k_vprintf  (const char *restrict format, va_list arg);
1350022 int     k_vsprintf (char *restrict string, const char *restrict format,
1350023                     va_list arg);
1350024 void    k_perror   (const char *s);
1350025
1350026 int     k_kill     (pid_t pid, int sig);
```



```
1350027 int     k_open     (const char *file, int oflags, ...);
1350028 void    k_close    (int fd);
1350029 ssize_t k_read     (int fd, void *buffer, size_t count);
1350030 //---------------------------------------------------------------------
1350031
1350032 #endif
```

kernel/k_libc/k_clock.c

«

Si veda la sezione u0.5.

```
1360001 #include <kernel/k_libc.h>
1360002 //---------------------------------------------------------------------
1360003 extern clock_t _clock_ticks;    // uint32_t
1360004 //---------------------------------------------------------------------
1360005 clock_t
1360006 k_clock (void)
1360007 {
1360008     return (_clock_ticks);
1360009 }
```

kernel/k_libc/k_close.c

«

Si veda la sezione u0.5.

```
1370001 #include <kernel/k_libc.h>
1370002 #include <kernel/fs.h>
1370003 //---------------------------------------------------------------------
1370004 void
1370005 k_close (int fdn)
1370006 {
1370007     fd_close ((pid_t) 0, fdn);
1370008     return;
1370009 }
```

kernel/k_libc/k_exit.s

«

Si veda la sezione u0.5.

```
1380001 .global _k_exit
1380002 ;---------------------------------------------------------------------
1380003 .text
1380004 ;---------------------------------------------------------------------
1380005 .align 2
1380006 _k_exit:
1380007 halt:
1380008     hlt
1380009     jmp halt
```

kernel/k_libc/k_kill.c

«

Si veda la sezione u0.5.

```
1390001 #include <kernel/k_libc.h>
1390002 #include <kernel/proc.h>
1390003 //---------------------------------------------------------------------
1390004 int
1390005 k_kill (pid_t pid, int sig)
1390006 {
1390007     return (proc_sys_kill ((pid_t) 0, pid, sig));
1390008 }
```

kernel/k_libc/k_open.c

«

Si veda la sezione u0.5.

```
1400001 #include <kernel/k_libc.h>
1400002 #include <kernel/fs.h>
1400003 #include <stdarg.h>
1400004 #include <string.h>
1400005 #include <errno.h>
1400006 //---------------------------------------------------------------------
1400007 int
1400008 k_open (const char *file, int oflags, ...)
1400009 {
1400010     mode_t  mode;
1400011     va_list ap;
1400012     //
1400013     va_start (ap, oflags);
1400014     mode = va_arg (ap, mode_t);
1400015     //
1400016     if (file == NULL || strlen (file) == 0)
1400017       {
1400018         errset (EINVAL);        // Invalid argument.
1400019         return (-1);
1400020       }
1400021     return (fd_open ((pid_t) 0, file, oflags, mode));
1400022 }
```

kernel/k_libc/k_perror.c

«

Si veda la sezione u0.5.

```
#include <kernel/k_libc.h>
#include <errno.h>
//---------------------------------------------------------------------------
void
k_perror (const char *s)
{
    //
    // If errno is zero, there is nothing to show.
    //
    if (errno == 0)
      {
        return;
      }
    //
    // Show the string if there is one.
    //
    if (s != NULL && strlen (s) > 0)
      {
        k_printf ("%s: ", s);
      }
    //
    // Show the translated error.
    //
    if (errfn[0] != 0 && errln != 0)
      {
        k_printf ("[%s:%u:%i] %s\n",
                  errfn, errln, errno, strerror (errno));
      }
    else
      {
        k_printf ("[%i] %s\n", errno, strerror (errno));
      }
}
```

kernel/k_libc/k_printf.c

«

Si veda la sezione u0.5.

```
#include <stdarg.h>
#include <kernel/k_libc.h>
//---------------------------------------------------------------------------
int
k_printf  (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return k_vprintf (format, ap);
}
```

kernel/k_libc/k_puts.c

«

Si veda la sezione u0.5.

```
#include <sys/os16.h>
#include <kernel/devices.h>
#include <kernel/k_libc.h>
#include <string.h>
//---------------------------------------------------------------------------
int
k_puts (const char *string)
{
    dev_io ((pid_t) 0, DEV_TTY, DEV_WRITE, (off_t) 0, string,
            strlen (string), NULL);
    dev_io ((pid_t) 0, DEV_TTY, DEV_WRITE, (off_t) 0, "\n", 1, NULL);
    return 1;
}
```

kernel/k_libc/k_read.c

«

Si veda la sezione u0.5.

```
#include <kernel/k_libc.h>
#include <kernel/fs.h>
//---------------------------------------------------------------------------
ssize_t
k_read (int fdn, void *buffer, size_t count)
{
    int     eof;
    ssize_t size;
    //
    eof = 0;
    //
    while (1)
      {
        size += fd_read ((pid_t) 0, fdn, buffer, count, &eof);
        if (size != 0 || eof)
          {
            break;
          }
      }
    return (size);
}
```


kernel/k_libc/k_stime.c

«

Si veda la sezione u0.5.

```
#include <kernel/k_libc.h>
//---------------------------------------------------------------------------
extern time_t  _clock_seconds;  // uint32_t
//---------------------------------------------------------------------------
int
k_stime (time_t *timer)
{
    _clock_seconds = *timer;
    return (0);
}
```

kernel/k_libc/k_time.c

«

Si veda la sezione u0.5.

```
#include <kernel/k_libc.h>
#include <stddef.h>
//---------------------------------------------------------------------------
extern time_t  _clock_seconds;  // uint32_t
//---------------------------------------------------------------------------
time_t
k_time (time_t *timer)
{
    if (timer != NULL)
      {
        *timer = _clock_seconds;
      }
    return (_clock_seconds);
}
```

kernel/k_libc/k_vprintf.c

«

Si veda la sezione u0.5.

```
#include <sys/os16.h>
#include <kernel/devices.h>
#include <stdarg.h>
#include <kernel/k_libc.h>
#include <string.h>
//---------------------------------------------------------------------------
int
k_vprintf (const char *restrict format, va_list arg)
{
    char  string[BUFSIZ];
    int   ret;
    string[0] = 0;
    ret = k_vsprintf (string, format, arg);
    dev_io ((pid_t) 0, DEV_CONSOLE, DEV_WRITE, (off_t) 0, string,
            strlen (string), NULL);
    return ret;
}
```

kernel/k_libc/k_vsprintf.c

«

Si veda la sezione u0.5.

```
#include <stdarg.h>
#include <kernel/k_libc.h>
#include <stdio.h>
//---------------------------------------------------------------------------
int
k_vsprintf (char *restrict string, const char *restrict format,
          va_list arg)
{
    int   ret;
    ret = vsnprintf (string, BUFSIZ, format, arg);
    return ret;
}
```

os16: «kernel/main.h»

«

Si veda la sezione u0.6.

```
#ifndef _KERNEL_MAIN_H
#define _KERNEL_MAIN_H  1

#include <sys/types.h>

void  menu (void);
pid_t run  (char *path, char *argv[], char *envp[]);
int   main (int argc, char *argv[], char *envp[]);

#endif
```

kernel/main/build.h

«

Si veda la sezione u0.6.

```
#define BUILD_DATE "2010.07.26 16:33:58"
```

Si veda la sezione u0.2.

```
1510001  .extern _main
1510002  .global ___mkargv
1510003  ;------------------------------------------------------------------------
1510004  ; Please note that, at the beginning, all the segment registers are
1510005  ; the same: CS==DS==ES==SS. But the data segments (DS, ES, SS) are meant
1510006  ; to be separated from the code, and they starts at: CS + __segoff.
1510007  ; The label "__segoff" is replaced by the linker with a constant value
1510008  ; (inside the code) with the segment offset to add to CS; this way
1510009  ; it is possibile to find where DS and ES should start.
1510010  ;------------------------------------------------------------------------
1510011  ; The following statement says that the code will start at "startup"
1510012  ; label.
1510013  ;------------------------------------------------------------------------
1510014  entry startup
1510015  ;------------------------------------------------------------------------
1510016  .text
1510017  ;------------------------------------------------------------------------
1510018  startup:
1510019      ;
1510020      ; Jump after initial data.
1510021      ;
1510022      jmp startup_code
1510023      ;
1510024  filler:
1510025      ;
1510026      ; After four bytes, from the start, there is the
1510027      ; magic number and other data.
1510028      ;
1510029      .space (0x0004 - (filler - startup))
1510030  magic:
1510031      ;
1510032      ; Add to "/etc/magic" the following line:
1510033      ;
1510034      ; 4  quad  0x6B65726E6F733136  os16 kernel
1510035      ;
1510036      .data4 0x6F733136   ; os16
1510037      .data4 0x6B65726E   ; kern
1510038      ;
1510039  segoff:                ;
1510040      .data2 __segoff     ; These values, for a kernel image,
1510041  etext:                 ; are not used.
1510042      .data2 __etext      ;
1510043  edata:                 ;
1510044      .data2 __edata      ;
1510045  ebss:                  ;
1510046      .data2 __end        ;
1510047  stack_size:            ;
1510048      .data2 0x0000       ;
1510049      ;
1510050      ; At the next label, the work begins.
1510051      ;
1510052  .align 2
1510053  startup_code:
1510054      ;
1510055      ; Check where we are. If we are at segment 0x1000,
1510056      ; then move to 0x3000.
1510057      ;
1510058      mov cx, cs
1510059      xor cx, #0x1000
1510060      jcxz move_code_from_0x1000_to_0x3000
1510061      ;
1510062      ; Check where we are. If we are at segment 0x3000,
1510063      ; then move to 0x0050, preserving the IVT and the BDA.
1510064      ;
1510065      mov cx, cs
1510066      xor cx, #0x3000
1510067      jcxz move_code_from_0x3000_to_0x0050
1510068      ;
1510069      ; Check where we are. If we are at segment 0x1050,
1510070      ; then jump to the main code.
1510071      ;
1510072      mov cx, cs
1510073      xor cx, #0x1050
1510074      jcxz main_code
1510075      ;
1510076      ; Otherwise, just halt.
1510077      ;
1510078      hlt
1510079      jmp startup_code
1510080      ;
1510081  move_code_from_0x1000_to_0x3000:
1510082      ;
1510083      cld                 ; Clear direction flag.
1510084      mov   ax, #0x3000   ; Set ES as the destination segment.
1510085      mov   es, ax        ;
1510086      mov   ax, #0x1000   ; Set DS as the source segment.
1510087      mov   ds, ax        ;
1510088      ;
1510089      mov   cx, #0x8000   ; Move 32768 words = 65536 byte (64 Kibyte).
1510090      mov   si, #0x0000   ; DS:SI == Source pointer
1510091      mov   di, #0x0000   ; ES:DI == Destination pointer
1510092      rep
1510093      movsw               ; Copy the array of words
1510094      ;
1510095      mov   ax, #0x4000   ; Set ES as the destination segment.
1510096      mov   es, ax        ;
1510097      mov   ax, #0x2000   ; Set DS as the source segment.
1510098      mov   ds, ax        ;
```



```
1510099      ;
1510100      mov   cx, #0x8000   ; Move 32768 words = 65536 byte (64 Kibyte).
1510101      mov   si, #0x0000   ; DS:SI == Source pointer
1510102      mov   di, #0x0000   ; ES:DI == Destination pointer
1510103      rep
1510104      movsw               ; Copy the array of words
1510105      ;
1510106      jmp far #0x3000:#0x0000     ; Go to the new kernel copy.
1510107      ;
1510108  move_code_from_0x3000_to_0x0050:
1510109      cld                 ; Clear direction flag.
1510110      ;
1510111      ; Text (instructions) is moved at segment 0x1050 (address 0x10500).
1510112      ;
1510113      mov   ax, #0x1050   ; Set ES as the destination segment.
1510114      mov   es, ax        ;
1510115      mov   ax, #0x3000   ; Set DS as the source segment.
1510116      mov   ds, ax        ;
1510117      ;
1510118      mov   cx, #0x8000   ; Move 32768 words = 65536 byte (64 Kibyte).
1510119      mov   si, #0x0000   ; DS:SI == Source pointer
1510120      mov   di, #0x0000   ; ES:DI == Destination pointer
1510121      rep
1510122      movsw               ; Copy the array of words
1510123      ;
1510124      ; Data is moved at segment 0x0050 (address 0x00500), before the
1510125      ; text segment.
1510126      ;
1510127      mov   ax, #0x0050   ; Set ES as the destination segment.
1510128      mov   es, ax        ;
1510129      mov   ax, #0x3000   ; Calculate where is the data segment:
1510130      add   ax, #__segoff ; it is at 0x3000 + __segoff.
1510131      mov   ds, ax        ; Set DS as the source segment.
1510132      ;
1510133      mov   cx, #0x8000   ; Move 32768 words = 65536 byte (64 Kibyte).
1510134      mov   si, #0x0000   ; DS:SI == Source pointer
1510135      mov   di, #0x0000   ; ES:DI == Destination pointer
1510136      rep
1510137      movsw               ; Copy the array of words
1510138      ;
1510139      jmp far #0x1050:#0x0000 ; Go to the new kernel copy.
1510140      ;
1510141  ;------------------------------------------------------------------------
1510142  main_code:
1510143      ;
1510144      ; Fix data segments!
1510145      ;
1510146      mov  ax, #0x0050
1510147      mov  ds, ax
1510148      mov  ss, ax
1510149      mov  es, ax
1510150      ;
1510151      ; Fix SP at the kernel stack bottom: the effective stack pointer
1510152      ; value should be 0x10000, but only 0x0000 can be written. At the
1510153      ; first push SP reaches 0xFFFE.
1510154      ;
1510155      mov sp, #0x0000
1510156      ;
1510157      ; Reset flags.
1510158      ;
1510159      push #0
1510160      popf
1510161      cli
1510162      ;
1510163      ; Call C main function, after kernel relocation and segments set up.
1510164      ;
1510165      push #0     ; This zero means NULL (envp[][] == NULL)
1510166      push #0     ; This zero means NULL (argv[][] == NULL)
1510167      push #0     ; This other zero means no arguments.
1510168      call _main
1510169      add  sp, #2
1510170      add  sp, #2
1510171      add  sp, #2
1510172      ;
1510173  .align 2
1510174  halt:
1510175      ;
1510176      ; It will never come back from the _kmain() call, but just for extra
1510177      ; security, loop forever.
1510178      ;
1510179      hlt
1510180      jmp halt
1510181      ;
1510182  ;------------------------------------------------------------------------
1510183  .align 2
1510184  ___mkargv:      ; Symbol '___mkargv' is used by Bcc inside the function
1510185      ret         ; 'main()' and must be present for a successful
1510186                  ; compilation.
1510187  ;------------------------------------------------------------------------
1510188  .align 2
1510189  .data
1510190  ;
1510191  ;------------------------------------------------------------------------
1510192  .align 2
1510193  .bss
```

kernel/main/main.c

«

Si veda la sezione u0.6.

```
1520001  #include <kernel/main.h>
1520002  #include <errno.h>
1520003  #include <fcntl.h>
1520004  #include <kernel/main/build.h>
1520005  #include <kernel/diag.h>
1520006  #include <kernel/fs.h>
1520007  #include <kernel/ibm_i86.h>
1520008  #include <kernel/k_libc.h>
1520009  #include <kernel/proc.h>
1520010  #include <libgen.h>
1520011  #include <stdlib.h>
1520012  #include <sys/os16.h>
1520013  #include <sys/stat.h>
1520014  #include <sys/types.h>
1520015  #include <unistd.h>
1520016  //------------------------------------------------------------------------
1520017  int
1520018  main (int argc, char *argv[], char *envp[])
1520019  {
1520020      unsigned int key;
1520021      pid_t        pid;
1520022      char        *exec_argv[2];
1520023      int          status;
1520024      int          exit;
1520025      //
1520026      // Reset video and select the initial console.
1520027      //
1520028      tty_init ();
1520029      //
1520030      // Show compilation date and time.
1520031      //
1520032      k_printf ("os16 build %s ram %i Kibyte\n", BUILD_DATE, int12 ());
1520033      //
1520034      // Set up disk management.
1520035      //
1520036      dsk_setup ();
1520037      //
1520038      // Clear heap for diagnosis.
1520039      //
1520040      heap_clear ();
1520041      //
1520042      // Set up process management. Process set up need the file system
1520043      // root directory already available.
1520044      //
1520045      proc_init ();
1520046      //
1520047      // The kernel will run interactively.
1520048      //
1520049      menu ();
1520050      //
1520051      for (exit = 0; exit == 0;)
1520052        {
1520053          //
1520054          // While in kernel code, timer interrupt don't start the
1520055          // scheduler. The kernel must leave control to the scheduler
1520056          // via a null system call.
1520057          //
1520058          sys (SYS_0, NULL, 0);
1520059          //
1520060          // Back to work: read the keyboard from the TTY device.
1520061          //
1520062          dev_io ((pid_t) 0, DEV_TTY, DEV_READ, 0L, &key, 1, NULL);
1520063          //
1520064          // Depending on the key, do something.
1520065          //
1520066          if (key == 0)
1520067            {
1520068              //
1520069              // No key is ready in the buffer keyboard.
1520070              //
1520071              continue;
1520072            }
1520073          else
1520074            {
1520075              //
1520076              // Move back the cursor, so that next print will overwrite
1520077              // it.
1520078              //
1520079              k_printf ("\b");
1520080            }
1520081          //
1520082          // A key was pressed: start to check what it was.
1520083          //
1520084          switch (key)
1520085            {
1520086              case 'h':
1520087                  menu ();
1520088                  break;
1520089              case '1':
1520090                  k_kill ((pid_t) 1, SIGKILL);                  // init
1520091                  break;
1520092              case '2':
1520093              case '3':
1520094              case '4':
1520095              case '5':
1520096              case '6':
1520097              case '7':
1520098              case '8':
1520099              case '9':
1520100                  k_kill ((pid_t) (key - '0'),  SIGTERM);        // others
1520101                  break;
1520102              case 'A':
1520103              case 'B':
1520104              case 'C':
1520105              case 'D':
1520106              case 'E':
1520107              case 'F':
1520108                  k_kill ((pid_t) (key - 'A' + 10),  SIGTERM);   // others
1520109                  break;
1520110              case 'a':
1520111                  run ("/bin/aaa", NULL, NULL);
1520112                  break;
1520113              case 'b':
1520114                  run ("/bin/bbb", NULL, NULL);
1520115                  break;
1520116              case 'c':
1520117                  run ("/bin/ccc", NULL, NULL);
1520118                  break;
1520119              case 'f':
1520120                  print_file_list ();
1520121                  break;
1520122              case 'm':
1520123                  status = path_mount ((uid_t) 0, "/dev/dsk1", "/usr",
1520124                                       MOUNT_DEFAULT);
1520125                  if (status < 0)
1520126                    {
1520127                      k_perror (NULL);
1520128                    }
1520129                  break;
1520130              case 'M':
1520131                  status = path_umount ((uid_t) 0, "/usr");
1520132                  if (status < 0)
1520133                    {
1520134                      k_perror (NULL);
1520135                    }
1520136                  break;
1520137              case 'n':
1520138                  print_inode_list ();
1520139                  break;
1520140              case 'N':
1520141                  print_inode_zones_list ();
1520142                  break;
1520143              case 'l':
1520144                  k_kill ((pid_t) 1, SIGCHLD);
1520145                  break;
1520146              case 'p':
1520147                  k_printf ("\n");
1520148                  print_proc_list ();
1520149                  print_segments ();
1520150                  k_printf (" ");
1520151                  print_kmem ();
1520152                  k_printf (" ");
1520153                  print_time ();
1520154                  k_printf ("\n");
1520155                  print_mb_map ();
1520156                  k_printf ("\n");
1520157                  break;
1520158              case 'x':
1520159                  exit = 1;
1520160                  break;
1520161              case 'q':
1520162                  k_printf ("System halted!\n");
1520163                  return (0);
1520164                  break;
1520165            }
1520166        }
1520167      //
1520168      // Load init.
1520169      //
1520170      exec_argv[0] = "/bin/init";
1520171      exec_argv[1] = NULL;
1520172      pid = run ("/bin/init", exec_argv, NULL);
1520173      //
1520174      // Just sleep.
1520175      //
1520176      while (1)
1520177        {
1520178          sys (SYS_0, NULL, 0);
1520179        }
1520180      //
1520181      k_printf ("System halted!\n");
1520182      return (0);
1520183  }
```


kernel/main/menu.c

«

Si veda la sezione u0.6.

```
1530001  #include <kernel/main.h>
1530002  #include <kernel/k_libc.h>
1530003  //------------------------------------------------------------------------
1530004  void
1530005  menu (void)
1530006  {
1530007      k_printf (
1530008  "+----------------------------------------------------------------+\n"
1530009  "| [h]      show this menu                                        |\n"
1530010  "| [p]      process status and memory map                         |\n"
1530011  "| [1]..[9] kill process  1 to 9                                  |\n"
```

```
"| [A]..[F] kill process 10 to 15                                    |\n"
"| [l]      send SIGCHLD to process 1                                |\n"
"| [a]..[c] run programs '/bin/aaa' to '/bin/ccc' in parallel        |\n"
"| [f]      system file status                                       |\n"
"| [n], [N] list of active inodes                                    |\n"
"| [m], [M] mount/umount '/dev/dsk1' at '/usr/'                      |\n"
"| [x]      exit interaction with kernel and start '/bin/init'       |\n"
"| [q]      quit kernel                                              |\n"
"'-----------------------------------------------------------------'\n"
);

}
```

kernel/main/run.c

«

Si veda la sezione u0.6.

```
#include <kernel/main.h>
#include <kernel/proc.h>
#include <kernel/k_libc.h>
#include <unistd.h>
//---------------------------------------------------------------------
pid_t
run (char *path, char *argv[], char *envp[])
{
    pid_t pid;
    //
    pid = fork ();
    if (pid == -1)
      {
        k_perror (NULL);
      }
    else if (pid == 0)
      {
        execve (path, argv, envp);
        k_perror (NULL);
        _exit (0);
      }
    return (pid);
}

```

os16: «kernel/memory.h»

«

Si veda la sezione u0.7.

```
#ifndef _KERNEL_MEMORY_H
#define _KERNEL_MEMORY_H       1

#include <stdint.h>
#include <stddef.h>
#include <sys/types.h>
//---------------------------------------------------------------------
#define MEM_BLOCK_SIZE     256  // 0x0100
#define MEM_MAX_BLOCKS    2560  // 655360/256 = 0xA0000/0x0100 = 0x0A00

extern uint16_t mb_table[MEM_MAX_BLOCKS/16]; // Memory blocks map.
//---------------------------------------------------------------------
typedef unsigned long int addr_t;
typedef unsigned int      segment_t;
typedef unsigned int      offset_t;
//---------------------------------------------------------------------
typedef struct {
    addr_t    address;
    segment_t segment;
    size_t    size;
} memory_t;
//---------------------------------------------------------------------
addr_t    address       (segment_t segment, offset_t offset);
//---------------------------------------------------------------------
uint16_t *mb_reference  (void);
ssize_t   mb_alloc      (addr_t address, size_t size);
void      mb_free       (addr_t address, size_t size);
int       mb_alloc_size (size_t size, memory_t *allocated);
//---------------------------------------------------------------------
void      mem_copy      (addr_t orig, addr_t dest, size_t size);
size_t    mem_read      (addr_t start, void *buffer, size_t size);
size_t    mem_write     (addr_t start, void *buffer, size_t size);
//---------------------------------------------------------------------

#endif
```

kernel/memory/address.c

«

Si veda la sezione u0.7.

```
#include <kernel/memory.h>
//---------------------------------------------------------------------
addr_t
address (segment_t segment, offset_t offset)
{
    addr_t a;
    a  = segment;
    a *= 16;
    a += offset;
    return (a);
}

```



```

```

kernel/memory/mb_alloc.c

«

Si veda la sezione u0.7.

```
#include <kernel/memory.h>
#include <kernel/ibm_i86.h>
#include <sys/os16.h>
#include <kernel/k_libc.h>
//---------------------------------------------------------------------
static int mb_block_set1   (int block);
//---------------------------------------------------------------------
ssize_t
mb_alloc (addr_t address, size_t size)
{
    unsigned int bstart;
    unsigned int bsize;
    unsigned int bend;
    unsigned int i;
    ssize_t      allocated = 0;
    addr_t       block_address;

    if (size == 0)
      {
        //
        // Zero means the maximum size.
        //
        bsize  = 0x10000L / MEM_BLOCK_SIZE;
      }
    else
      {
        bsize  = size / MEM_BLOCK_SIZE;
      }

    bstart = address / MEM_BLOCK_SIZE;

    if (size % MEM_BLOCK_SIZE)
      {
        bend = bstart + bsize;
      }
    else
      {
        bend = bstart + bsize - 1;
      }

    for (i = bstart; i <= bend; i++)
      {
        if (mb_block_set1 (i))
          {
            allocated += MEM_BLOCK_SIZE;
          }
        else
          {
            block_address = i;
            block_address *= MEM_BLOCK_SIZE;
            k_printf ("Kernel alert: mem block %04x, at address ", i);
            k_printf ("%05lx, already allocated!\n", block_address);
            break;
          }
      }
    return (allocated);
}

//---------------------------------------------------------------------
static int
mb_block_set1 (int block)
{
    int i         = block / 16;
    int j         = block % 16;
    uint16_t mask = 0x8000 >> j;
    if (mb_table[i] & mask)
      {
        return (0);     // The block is already set to 1 inside the map!
      }
    else
      {
        mb_table[i] = mb_table[i] | mask;
        return (1);
      }
}
```

kernel/memory/mb_alloc_size.c

«

Si veda la sezione u0.7.

```
#include <kernel/memory.h>
#include <kernel/ibm_i86.h>
#include <sys/os16.h>
#include <errno.h>
//---------------------------------------------------------------------
static int mb_block_status (int block);
//---------------------------------------------------------------------
```

```
1580008 int
1580009 mb_alloc_size (size_t size, memory_t *allocated)
1580010 {
1580011     unsigned int bsize;
1580012     unsigned int i;
1580013     unsigned int j;
1580014     unsigned int found = 0;
1580015     addr_t       alloc_addr;
1580016     ssize_t      alloc_size;
1580017
1580018     if (size == 0)
1580019       {
1580020         //
1580021         // Zero means the maximum size.
1580022         //
1580023         bsize = 0x10000L / MEM_BLOCK_SIZE;
1580024       }
1580025     else if (size % MEM_BLOCK_SIZE)
1580026       {
1580027         bsize = size / MEM_BLOCK_SIZE + 1;
1580028       }
1580029     else
1580030       {
1580031         bsize = size / MEM_BLOCK_SIZE;
1580032       }
1580033
1580034     for (i = 0; i < (MEM_MAX_BLOCKS - bsize) && !found; i++)
1580035       {
1580036         for (j = 0; j < bsize; j++)
1580037           {
1580038             found = !mb_block_status (i+j);
1580039             if (!found)
1580040               {
1580041                 i += j;
1580042                 break;
1580043               }
1580044           }
1580045       }
1580046
1580047     if (found && (j == bsize))
1580048       {
1580049         alloc_addr  = i - 1;
1580050         alloc_addr *= MEM_BLOCK_SIZE;
1580051         alloc_size  = bsize * MEM_BLOCK_SIZE;
1580052         alloc_size  = mb_alloc (alloc_addr, (size_t) alloc_size);
1580053         if (alloc_size <= 0)
1580054           {
1580055             errset (ENOMEM);
1580056             return (-1);
1580057           }
1580058         else if (alloc_size < size)
1580059           {
1580060             mb_free (alloc_addr, (size_t) alloc_size);
1580061             errset (ENOMEM);
1580062             return (-1);
1580063           }
1580064         else
1580065           {
1580066             allocated->address = alloc_addr;
1580067             allocated->segment = alloc_addr / 16;
1580068             allocated->size    = (size_t) alloc_size;
1580069           }
1580070         return (0);
1580071       }
1580072     else
1580073      {
1580074         errset (ENOMEM);
1580075         return (-1);
1580076      }
1580077 }
1580078 //-------------------------------------------------------------------------
1580079 static int
1580080 mb_block_status (int block)
1580081 {
1580082     int i         = block / 16;
1580083     int j         = block % 16;
1580084     uint16_t mask = 0x8000 >> j;
1580085     return ((int) (mb_table[i] & mask));
1580086 }
```

kernel/memory/mb_free.c

«

Si veda la sezione u0.7.

```
1590001 #include <kernel/memory.h>
1590002 #include <kernel/ibm_i86.h>
1590003 #include <sys/os16.h>
1590004 #include <kernel/k_libc.h>
1590005 //-------------------------------------------------------------------------
1590006 static int mb_block_set0   (int block);
1590007 //-------------------------------------------------------------------------
1590008 void
1590009 mb_free (addr_t address, size_t size)
1590010 {
1590011     unsigned int bstart;
1590012     unsigned int bsize;
1590013     unsigned int bend;
1590014     unsigned int i;
1590015     addr_t       block_address;
1590016     if (size == 0)
1590017       {
```



```
1590018         //
1590019         // Zero means the maximum size.
1590020         //
1590021         bsize  = 0x10000L / MEM_BLOCK_SIZE;
1590022       }
1590023     else
1590024       {
1590025         bsize  = size / MEM_BLOCK_SIZE;
1590026       }
1590027
1590028     bstart = address / MEM_BLOCK_SIZE;
1590029
1590030     if (size % MEM_BLOCK_SIZE)
1590031       {
1590032         bend = bstart + bsize;
1590033       }
1590034     else
1590035       {
1590036         bend = bstart + bsize - 1;
1590037       }
1590038
1590039     for (i = bstart; i <= bend; i++)
1590040       {
1590041         if (mb_block_set0 (i))
1590042           {
1590043             ;
1590044           }
1590045         else
1590046           {
1590047             block_address = i;
1590048             block_address *= MEM_BLOCK_SIZE;
1590049             k_printf ("Kernel alert: mem block %04x, at address ", i);
1590050             k_printf ("%05lx, already released!\n", block_address);
1590051           }
1590052       }
1590053 }
1590054 //-------------------------------------------------------------------------
1590055 static int
1590056 mb_block_set0 (int block)
1590057 {
1590058     int i         = block / 16;
1590059     int j         = block % 16;
1590060     uint16_t mask = 0x8000 >> j;
1590061     if (mb_table[i] & mask)
1590062       {
1590063         mb_table[i] = mb_table[i] & ~mask;
1590064         return (1);
1590065       }
1590066     else
1590067       {
1590068         return (0);     // The block is already set to 0 inside the map!
1590069       }
1590070 }
```

kernel/memory/mb_reference.c

«

Si veda la sezione u0.7.

```
1600001 #include <stdint.h>
1600002 #include <kernel/memory.h>
1600003 //-------------------------------------------------------------------------
1600004 uint16_t *
1600005 mb_reference (void)
1600006 {
1600007     return mb_table;
1600008 }
1600009
```

kernel/memory/mb_table.c

«

Si veda la sezione u0.7.

```
1610001 #include <kernel/memory.h>
1610002 #include <stdint.h>
1610003 //-------------------------------------------------------------------------
1610004 uint16_t mb_table[MEM_MAX_BLOCKS/16]; // Memory blocks map.
1610005 //-------------------------------------------------------------------------
```

kernel/memory/mem_copy.c

«

Si veda la sezione u0.7.

```
1620001 #include <kernel/memory.h>
1620002 #include <kernel/ibm_i86.h>
1620003 #include <sys/os16.h>
1620004 //-------------------------------------------------------------------------
1620005 void
1620006 mem_copy (addr_t orig, addr_t dest, size_t size)
1620007 {
1620008     segment_t seg_orig = orig / 16;
1620009     offset_t  off_orig = orig % 16;
1620010     segment_t seg_dest = dest / 16;
1620011     offset_t  off_dest = dest % 16;
1620012     ram_copy (seg_orig, off_orig, seg_dest, off_dest, size);
1620013 }
```

kernel/memory/mem_read.c

«

Si veda la sezione u0.7.

```
1630001 #include <kernel/memory.h>
1630002 #include <kernel/ibm_i86.h>
1630003 #include <sys/os16.h>
1630004 //-----------------------------------------------------------------------
1630005 size_t
1630006 mem_read (addr_t start, void *buffer, size_t size)
1630007 {
1630008     unsigned int      segment = start / 16;
1630009     unsigned int      offset  = start % 16;
1630010     unsigned long int end;
1630011     end  = start;
1630012     end += size;
1630013     if (end > 0x000FFFFFL)
1630014       {
1630015         size = 0x000FFFFFL - start;
1630016       }
1630017     ram_copy (segment, offset, seg_d (), (unsigned int) buffer, size);
1630018     return (size);
1630019 }
```

kernel/memory/mem_write.c

«

Si veda la sezione u0.7.

```
1640001 #include <kernel/memory.h>
1640002 #include <kernel/ibm_i86.h>
1640003 #include <sys/os16.h>
1640004 //-----------------------------------------------------------------------
1640005 size_t
1640006 mem_write (addr_t start, void *buffer, size_t size)
1640007 {
1640008     unsigned int      segment = start / 16;
1640009     unsigned int      offset  = start % 16;
1640010     unsigned long int end;
1640011     end  = start;
1640012     end += size;
1640013     if (end > 0x000FFFFFL)
1640014       {
1640015         size = 0x000FFFFFL - start;
1640016       }
1640017     ram_copy (seg_d (), (unsigned int) buffer, segment, offset, size);
1640018     return (size);
1640019 }
```

os16: «kernel/proc.h»

«

Si veda la sezione u0.8.

```
1650001 #ifndef _KERNEL_PROC_H
1650002 #define _KERNEL_PROC_H  1
1650003
1650004 #include <kernel/devices.h>
1650005 #include <kernel/memory.h>
1650006 #include <kernel/fs.h>
1650007 #include <kernel/tty.h>
1650008 #include <sys/types.h>
1650009 #include <sys/stat.h>
1650010 #include <sys/os16.h>
1650011 #include <stddef.h>
1650012 #include <stdint.h>
1650013 #include <time.h>
1650014
1650015 //-----------------------------------------------------------------------
1650016 #define CLOCK_FREQUENCY_DIVISOR     65535 // [1]
1650017 //
1650018 // [1]
1650019 // Internal clock frequency is (3579545/3) Hz.
1650020 // This value is divided by 65535 (0xFFFF) giving 18.2 Hz.
1650021 // The divisor value, 65535, if fixed!
1650022 //
1650023 //-----------------------------------------------------------------------
1650024 #define PROC_EMPTY                      0
1650025 #define PROC_CREATED                    1
1650026 #define PROC_READY                      2
1650027 #define PROC_RUNNING                    3
1650028 #define PROC_SLEEPING                   4
1650029 #define PROC_ZOMBIE                     5
1650030 //-----------------------------------------------------------------------
1650031 #define MAGIC_OS16              0x6F733136L  // os16
1650032 #define MAGIC_OS16_APPL         0x6170706CL  // appl
1650033 #define MAGIC_OS16_KERN         0x6B65726EL  // kern
1650034 //-----------------------------------------------------------------------
1650035 #define PROCESS_MAX  16                 // Process slots.
1650036
1650037 typedef struct {
1650038     pid_t             ppid;             // Parent PID.
1650039     pid_t             pgrp;             // Process group ID.
1650040     uid_t             uid;              // Real user ID
1650041     uid_t             euid;             // Effective user ID.
1650042     uid_t             suid;             // Saved user ID.
1650043     dev_t             device_tty;       // Controlling terminal.
1650044     char              path_cwd[PATH_MAX];
1650045                                         // Working directory path.
1650046     inode_t          *inode_cwd;        // Working directory inode.
1650047     int               umask;            // File creation mask.
1650048     unsigned long int sig_status;       // Active signals.
1650049     unsigned long int sig_ignore;       // Signals to be ignored.
1650050     clock_t           usage;            // Clock ticks CPU time usage.
1650051     unsigned int      status;
1650052     int               wakeup_events;    // Wake up for something.
1650053     int               wakeup_signal;    // Signal waited.
1650054     unsigned int      wakeup_timer;     // Seconds to wait for.
1650055     addr_t            address_i;
1650056     segment_t         segment_i;
1650057     size_t            size_i;
1650058     addr_t            address_d;
1650059     segment_t         segment_d;
1650060     size_t            size_d;
1650061     uint16_t          sp;
1650062     int               ret;
1650063     char              name[PATH_MAX];
1650064     fd_t              fd[FOPEN_MAX];
1650065 } proc_t;
1650066
1650067 extern proc_t   proc_table[PROCESS_MAX];
1650068 //-----------------------------------------------------------------------
1650069 typedef struct {
1650070     uint32_t filler0;
1650071     uint32_t magic0;
1650072     uint32_t magic1;
1650073     uint16_t segoff;
1650074     uint16_t etext;
1650075     uint16_t edata;
1650076     uint16_t ebss;
1650077     uint16_t ssize;
1650078 } header_t;
1650079 //-----------------------------------------------------------------------
1650080 void         _ivt_load          (void);
1650081 #define      ivt_load()         (_ivt_load ())
1650082 void         proc_init          (void);
1650083 void         proc_scheduler     (uint16_t *sp, segment_t *segment);
1650084 void         sysroutine         (uint16_t *sp, segment_t *segment,
1650085                                  uint16_t syscallnr, uint16_t msg_off,
1650086                                  uint16_t msg_size);
1650087 proc_t      *proc_reference     (pid_t pid);
1650088 //-----------------------------------------------------------------------
1650089 int          proc_sys_exec      (uint16_t *sp, segment_t *segment_d,
1650090                                  pid_t pid, const char *path,
1650091                                  unsigned int argc, char *arg_data,
1650092                                  unsigned int envc, char *env_data);
1650093 void         proc_sys_exit      (pid_t pid, int status);
1650094 pid_t        proc_sys_fork      (pid_t ppid, uint16_t sp);
1650095 int          proc_sys_kill      (pid_t pid_killer, pid_t pid_target,
1650096                                  int sig);
1650097 int          proc_sys_seteuid   (pid_t pid, uid_t euid);
1650098 int          proc_sys_setuid    (pid_t pid, uid_t uid);
1650099 sighandler_t proc_sys_signal    (pid_t pid, int sig,
1650100                                  sighandler_t handler);
1650101 pid_t        proc_sys_wait      (pid_t pid, int *status);
1650102 //-----------------------------------------------------------------------
1650103 void         proc_dump_memory   (pid_t pid, addr_t address,
1650104                                  size_t size, char *name);
1650105 void         proc_available     (pid_t pid);
1650106 pid_t        proc_find          (segment_t segment_d);
1650107 void         proc_sch_signals   (void);
1650108 void         proc_sch_terminals (void);
1650109 void         proc_sch_timers    (void);
1650110 void         proc_sig_chld      (pid_t parent, int sig);
1650111 void         proc_sig_cont      (pid_t pid, int sig);
1650112 void         proc_sig_core      (pid_t pid, int sig);
1650113 int          proc_sig_ignore    (pid_t pid, int sig);
1650114 void         proc_sig_off       (pid_t pid, int sig);
1650115 void         proc_sig_on        (pid_t pid, int sig);
1650116 int          proc_sig_status    (pid_t pid, int sig);
1650117 void         proc_sig_stop      (pid_t pid, int sig);
1650118 void         proc_sig_term      (pid_t pid, int sig);
1650119
1650120 #endif
```

kernel/proc/_isr.s

«

Si veda la sezione i159.8.1.

```
1660001 .extern _proc_scheduler
1660002 .extern _sysroutine
1660003 .global __ksp
1660004 .global __clock_ticks
1660005 .global __clock_seconds
1660006 .global isr_1C
1660007 .global isr_80
1660008 ;-------------------------------------------------------------------------
1660009 ; The kernel code segment starts at 0x10500 (segment 0x1050).
1660010 ; The kernel data segments start at 0x00500 (segment 0x0050).
1660011 ; To switch to the kernel data segments, DS, ES and SS are set to
1660012 ; 0x0050. To identify the kernel context, the DS register is checked:
1660013 ; if it is equal to 0x0050, it is the kernel.
1660014 ;-------------------------------------------------------------------------
1660015 .data
1660016 ;-------------------------------------------------------------------------
1660017 .align 2
1660018 proc_ss_0:       .word 0x0000
1660019 proc_sp_0:       .word 0x0000
1660020 proc_ss_1:       .word 0x0000
1660021 proc_sp_1:       .word 0x0000
1660022 proc_syscallnr:  .word 0x0000
1660023 proc_msg_offset: .word 0x0000
1660024 proc_msg_size:   .word 0x0000
```

```
__ksp:           .word 0x0000
__clock_ticks:
ticks_lo:        .word 0x0000
ticks_hi:        .word 0x0000
__clock_seconds:
seconds_lo:      .word 0x0000
seconds_hi:      .word 0x0000
;--------------------------------------------------------------------------
.text
;--------------------------------------------------------------------------
; IRQ 0: "timer".
; IRQ 0 is associated to INT 8, and after the BIOS work is done,
; INT 1C is called. Standard INT 1C has nothing to do, but is
; useful to call extra work for the timer. As the original BIOS
; interrupts are used, the INT 1C is reprogrammed, keeping intact
; the Standard INT 8.
;--------------------------------------------------------------------------
.align 2
isr_1C:
    ;--------------------------------------------------------------
    ; Inside the process stack, the CPU already put:
    ;
    ;       [omissis]
    ;       push flags
    ;       push cs
    ;       push ip
    ;--------------------------------------------------------------
    ;
    ; Save into process stack:
    ;
    push   es   ; extra segment
    push   ds   ; data segment
    push   di   ; destination index
    push   si   ; source index
    push   bp   ; base pointer
    push   bx   ; BX
    push   dx   ; DX
    push   cx   ; CX
    push   ax   ; AX
    ;
    ; Set the data segments to the kernel data area,
    ; so that the following variables can be accessed.
    ;
    mov ax, #0x0050  ; DS and ES.
    mov ds, ax       ;
    mov es, ax       ;
    ;
    ; Increment time counters, to keep time.
    ;
    add ticks_lo, #1    ; Clock ticks counter.
    adc ticks_hi, #0    ;
    ;
    mov dx, ticks_hi    ;
    mov ax, ticks_lo    ; DX := ticks % 18
    mov cx, #18         ;
    div cx              ;
    mov ax, #0          ; If the ticks value can be divided by 18,
    cmp ax, dx          ; the seconds is incremented by 1.
    jnz L1              ;
    add seconds_lo, #1  ;
    adc seconds_hi, #0  ;
    ;
L1: ; Save process stack registers into kernel data segment.
    ;
    mov proc_ss_0, ss   ; Save process stack segment.
    mov proc_sp_0, sp   ; Save process stack pointer.
    ;
    ; Check if it is already in kernel mode.
    ;
    mov dx, proc_ss_0
    mov ax, #0x0050     ; Kernel data area.
    cmp dx, ax
    je L2
    ;
    ; If we are here, a user process was interrupted.
    ; Switch to the kernel stack.
    ;
    mov ax, #0x0050     ; Kernel data area.
    mov ss, ax
    mov sp, __ksp
    ;
    ; Call the scheduler.
    ;
    push #proc_ss_0     ; &proc_ss_0
    push #proc_sp_0     ; &proc_sp_0
    call _proc_scheduler
    add  sp, #2
    add  sp, #2
    ;
    ; Restore process stack registers from kernel data segment.
    ;
    mov ss, proc_ss_0   ; Restore process stack segment.
    mov sp, proc_sp_0   ; Restore process stack pointer.
    ;
L2: ; Restore from process stack:
    ;
    pop    ax
    pop    cx
    pop    dx
    pop    bx
    pop    bp
```



```
    pop    si
    pop    di
    pop    ds
    pop    es
    ;
    ; Return from interrupt: will restore CS:IP and FLAGS
    ; from process stack.
    ;
    iret
;--------------------------------------------------------------------------
; Syscall.
;--------------------------------------------------------------------------
.align 2
isr_80:
    ;--------------------------------------------------------------
    ; Inside the process stack, we already have:
    ;       push #message_size
    ;       push &message_structure     ; the relative address of it
    ;       push #syscall_number
    ;       push #back_address  ; made by a call to _sys function
    ;       push flags          ; made by int #0x80
    ;       push cs             ; made by int #0x80
    ;       push ip             ; made by int #0x80
    ;--------------------------------------------------------------
    ;
    ; Save into process stack:
    ;
    push   es   ; extra segment
    push   ds   ; data segment
    push   di   ; destination index
    push   si   ; source index
    push   bp   ; base pointer
    push   bx   ; BX
    push   dx   ; DX
    push   cx   ; CX
    push   ax   ; AX
    ;
    ; Set the data segments to the kernel data area,
    ; so that the following variables can be accessed.
    ;
    mov ax, #0x0050     ; DS and ES.
    mov ds, ax          ;
    mov es, ax          ;
    ;
    ; Save process stack registers into kernel data segment.
    ;
    mov proc_ss_1, ss   ; Save process stack segment.
    mov proc_sp_1, sp   ; Save process stack pointer.
    ;
    ; Save some more data, from the system call.
    ;
    mov bp, sp
    mov ax, +26[bp]
    mov proc_syscallnr,  ax
    mov ax, +28[bp]
    mov proc_msg_offset, ax
    mov ax, +30[bp]
    mov proc_msg_size,   ax
    ;
    ; Check if it is already the kernel stack.
    ;
    mov dx, ss
    mov ax, #0x0050     ; Kernel data area.
    cmp dx, ax
    jne L3
    ;
    ; It is already the kernel stack, so, the variable "_ksp" is
    ; aligned to current stack pointer. This way, the first syscall
    ; can work without having to set the "_ksp" variable to some
    ; reasonable value.
    ;
    mov __ksp, sp
    ;
L3: ; Switch to the kernel stack.
    ;
    mov ax, #0x0050     ; Kernel data area.
    mov ss, ax
    mov sp, __ksp
    ;
    ; Call the external hardware interrupt handler.
    ;
    push proc_msg_size
    push proc_msg_offset
    push proc_syscallnr
    push #proc_ss_1     ; &proc_ss_1
    push #proc_sp_1     ; &proc_sp_1
    call _sysroutine
    add  sp, #2
    add  sp, #2
    add  sp, #2
    add  sp, #2
    add  sp, #2
    ;
    ; Restore process stack registers from kernel data segment.
    ;
    mov ss, proc_ss_1   ; Restore process stack segment.
    mov sp, proc_sp_1   ; Restore process stack pointer.
    ;
    ; Restore from process stack:
    ;
    pop    ax
```

```
1660227     pop    cx
1660228     pop    dx
1660229     pop    bx
1660230     pop    bp
1660231     pop    si
1660232     pop    di
1660233     pop    ds
1660234     pop    es
1660235     ;
1660236     ; Return from interrupt: will restore CS:IP and FLAGS
1660237     ; from process stack.
1660238     ;
1660239     iret
```

kernel/proc/_ivt_load.s

«

Si veda la sezione i159.8.2.

```
1670001 .extern isr_1C
1670002 .extern isr_80
1670003 .global __ivt_load
1670004 ;------------------------------------------------------------------------
1670005 .text
1670006 ;------------------------------------------------------------------------
1670007 ; Load IVT.
1670008 ;
1670009 ; Currently, only the timer function and the syscall are loaded.
1670010 ;------------------------------------------------------------------------
1670011 .align 2
1670012 __ivt_load:
1670013     enter #0, #0                ; No local variables.
1670014     pushf
1670015     cli
1670016     pusha
1670017     ;
1670018     mov   ax,   #0              ; Change the DS segment to 0.
1670019     mov   ds,   ax              ;
1670020     ;
1670021     mov   bx,   #112            ; Timer         INT 0x08 (8) --> 0x1C
1670022     mov   [bx], #isr_1C         ; offset
1670023     mov   bx,   #114            ;
1670024     mov   [bx], cs              ; segment
1670025     ;
1670026     mov   bx,   #512            ; Syscall       INT 0x80 (128)
1670027     mov   [bx], #isr_80         ; offset
1670028     mov   bx,   #514            ;
1670029     mov   [bx], cs              ; segment
1670030     ;
1670031     mov   ax,   #0x0050         ; Put the DS segment back to the right
1670032     mov   ds,   ax              ; value.
1670033     ;
1670034     ;
1670035     ;
1670036     popa
1670037     popf
1670038     leave
1670039     ret
```

kernel/proc/proc_available.c

«

Si veda la sezione i159.8.3.

```
1680001 #include <kernel/proc.h>
1680002 //---------------------------------------------------------------------
1680003 void
1680004 proc_available (pid_t pid)
1680005 {
1680006     proc_table[pid].ppid          = -1;
1680007     proc_table[pid].pgrp          = -1;
1680008     proc_table[pid].uid           = -1;
1680009     proc_table[pid].euid          = -1;
1680010     proc_table[pid].suid          = -1;
1680011     proc_table[pid].sig_status    = 0;
1680012     proc_table[pid].sig_ignore    = 0;
1680013     proc_table[pid].usage         = 0;
1680014     proc_table[pid].status        = PROC_EMPTY;
1680015     proc_table[pid].wakeup_events = 0;
1680016     proc_table[pid].wakeup_signal = 0;
1680017     proc_table[pid].wakeup_timer  = 0;
1680018     proc_table[pid].segment_i     = -1;
1680019     proc_table[pid].address_i     = -1L;
1680020     proc_table[pid].size_i        = -1;
1680021     proc_table[pid].segment_d     = -1;
1680022     proc_table[pid].address_d     = -1L;
1680023     proc_table[pid].size_d        = -1;
1680024     proc_table[pid].sp            = 0;
1680025     proc_table[pid].ret           = 0;
1680026     proc_table[pid].inode_cwd     = 0;
1680027     proc_table[pid].path_cwd[0]   = 0;
1680028     proc_table[pid].umask         = 0;
1680029     proc_table[pid].name[0]       = 0;
1680030 }
```


kernel/proc/proc_dump_memory.c

«

Si veda la sezione i159.8.4.

```
1690001 #include <kernel/proc.h>
1690002 #include <fcntl.h>
1690003 //---------------------------------------------------------------------
1690004 void
1690005 proc_dump_memory (pid_t pid, addr_t address, size_t size, char *name)
1690006 {
1690007     int     fdn;
1690008     char    buffer[SB_BLOCK_SIZE];
1690009     ssize_t size_written;
1690010     ssize_t size_written_total;
1690011     ssize_t size_read;
1690012     ssize_t size_read_total;
1690013     ssize_t size_total;
1690014     //
1690015     // Dump the code segment to disk.
1690016     //
1690017     fdn = fd_open (pid, name, (O_WRONLY|O_CREAT|O_TRUNC),
1690018                    (mode_t) (S_IFREG|00644));
1690019     if (fdn < 0)
1690020       {
1690021         //
1690022         // There is a problem: just let it go.
1690023         //
1690024         return;
1690025       }
1690026     //
1690027     // Fix size: (size_t) 0 is equivalent to (ssize_t) 0x10000.
1690028     //
1690029     size_total = size;
1690030     if (size_total == 0)
1690031       {
1690032         size_total = 0x10000;
1690033       }
1690034     //
1690035     // Read the memory and write it to disk.
1690036     //
1690037     for (size_read = 0, size_read_total = 0;
1690038          size_read_total < size_total;
1690039          size_read_total += size_read, address += size_read)
1690040       {
1690041         size_read = mem_read (address, buffer, SB_BLOCK_SIZE);
1690042         //
1690043         for (size_written = 0, size_written_total = 0;
1690044              size_written_total < size_read;
1690045              size_written_total += size_written)
1690046           {
1690047             size_written = fd_write (pid, fdn,
1690048                                      &buffer[size_written_total],
1690049                                      (size_t) (size_read - size_written_total));
1690050             //
1690051             if (size_written < 0)
1690052               {
1690053                 fd_close (pid, fdn);
1690054                 return;
1690055               }
1690056           }
1690057       }
1690058     fd_close (pid, fdn);
1690059 }
```

kernel/proc/proc_find.c

«

Si veda la sezione i159.8.5.

```
1700001 #include <kernel/proc.h>
1700002 #include <kernel/k_libc.h>
1700003 #include <kernel/diag.h>
1700004 //---------------------------------------------------------------------
1700005 pid_t
1700006 proc_find (segment_t segment_d)
1700007 {
1700008     int    pid;
1700009     addr_t address_d;
1700010     for (pid = 0; pid < PROCESS_MAX; pid++)
1700011       {
1700012         if (proc_table[pid].segment_d == segment_d)
1700013           {
1700014             break;
1700015           }
1700016       }
1700017     if (pid >= PROCESS_MAX)
1700018       {
1700019         address_d  = segment_d;
1700020         address_d *= 16;
1700021         k_printf ("\n"
1700022                   "Kernel panic: cannot find the interrupted process "
1700023                   "inside the process table. "
1700024                   "The wanted process has data segment 0x%04x \n"
1700025                   "(effective address %05lx)!\n",
1700026                   (unsigned int) segment_d, address_d);
1700027         print_proc_list ();
1700028         print_segments ();
1700029         k_printf (" ");
1700030         print_kmem ();
1700031         k_printf (" ");
1700032         print_time ();
1700033         k_printf ("\n");
```

```
1700034         print_mb_map ();
1700035         k_printf ("\n");
1700036         k_exit (0);
1700037       }
1700038     return (pid);
1700039 }
```

kernel/proc/proc_init.c

«

Si veda la sezione i159.8.6.

```
1710001 #include <kernel/proc.h>
1710002 #include <kernel/k_libc.h>
1710003 #include <string.h>
1710004 //---------------------------------------------------------------------
1710005 extern uint16_t _etext;
1710006 //---------------------------------------------------------------------
1710007 void
1710008 proc_init (void)
1710009 {
1710010     uint8_t  divisor_lo;
1710011     uint8_t  divisor_hi;
1710012     pid_t    pid;
1710013     int      fdn;                // File descriptor index;
1710014     addr_t   start;              // Used for effective memory addresses.
1710015     size_t   size;               // Used for memory allocation.
1710016     inode_t *inode;
1710017     sb_t    *sb;
1710018     //
1710019     // Clear interrupts (should already be cleared).
1710020     //
1710021     cli ();
1710022     //
1710023     // Load Interrupt vector table (IVT).
1710024     //
1710025     ivt_load ();
1710026     //
1710027     // Configure the clock: must be the original values, because
1710028     // the BIOS depends on it!
1710029     //
1710030     // Base frequency is 1193181 Hz and it should divided.
1710031     // Resulting frequency must be from 18.22 Hz and 1193181 Hz.
1710032     // The calculated value (the divisor) must be sent to the
1710033     // PIT (programmable interval timer), divided in two pieces.
1710034     //
1710035     divisor_lo = (CLOCK_FREQUENCY_DIVISOR & 0xFF);         // Low  byte.
1710036     divisor_hi = (CLOCK_FREQUENCY_DIVISOR / 0x100) & 0xFF; // High byte.
1710037     out_8 (0x43, 0x36);
1710038     out_8 (0x40, divisor_lo);        // Lower byte.
1710039     out_8 (0x40, divisor_hi);        // Higher byte.
1710040     //
1710041     // Set all memory reference to some invalid data.
1710042     //
1710043     for (pid = 0; pid < PROCESS_MAX; pid++)
1710044       {
1710045         proc_available (pid);
1710046       }
1710047     //
1710048     // Mount root file system.
1710049     //
1710050     inode = NULL;
1710051     sb = sb_mount (DEV_DSK0, &inode, MOUNT_DEFAULT);
1710052     if (sb == NULL || inode == NULL)
1710053       {
1710054         k_perror ("Kernel panic: cannot mount root file system:");
1710055         k_exit (0);
1710056       }
1710057     //
1710058     // Set up the process table with the kernel.
1710059     //
1710060     proc_table[0].ppid          = 0;
1710061     proc_table[0].pgrp          = 0;
1710062     proc_table[0].uid           = 0;
1710063     proc_table[0].euid          = 0;
1710064     proc_table[0].suid          = 0;
1710065     proc_table[0].device_tty    = DEV_UNDEFINED;
1710066     proc_table[0].sig_status    = 0;
1710067     proc_table[0].sig_ignore    = 0;
1710068     proc_table[0].usage         = 0;
1710069     proc_table[0].status        = PROC_RUNNING;
1710070     proc_table[0].wakeup_events = 0;
1710071     proc_table[0].wakeup_signal = 0;
1710072     proc_table[0].wakeup_timer  = 0;
1710073     proc_table[0].segment_i     = seg_i ();
1710074     proc_table[0].address_i     = seg_i ();
1710075     proc_table[0].address_i    *= 16;
1710076     proc_table[0].size_i        = (size_t) &_etext;
1710077     proc_table[0].segment_d     = seg_d ();
1710078     proc_table[0].address_d     = seg_d ();
1710079     proc_table[0].address_d    *= 16;
1710080     proc_table[0].size_d        = 0;     // Maximum size: 0x10000.
1710081     proc_table[0].sp            = 0;     // To be set at next interrupt.
1710082     proc_table[0].ret           = 0;
1710083     proc_table[0].umask         = 0022;  // Default umask.
1710084     proc_table[0].inode_cwd     = inode; // Root fs inode.
1710085     strncpy (proc_table[0].path_cwd, "/", PATH_MAX);
1710086     strncpy (proc_table[0].name, "os16 kernel", PATH_MAX);
1710087     //
1710088     // Ensure to have a terminated string.
1710089     //
1710090     proc_table[0].name[PATH_MAX-1] = 0;
```



```
1710091     //
1710092     // Reset file descriptors.
1710093     //
1710094     for (fdn = 0; fdn < OPEN_MAX; fdn++)
1710095       {
1710096         proc_table[0].fd[fdn].fl_flags = 0;
1710097         proc_table[0].fd[fdn].fd_flags = 0;
1710098         proc_table[0].fd[fdn].file     = NULL;
1710099       }
1710100     //
1710101     // Allocate memory for the code segment.
1710102     //
1710103     mb_alloc (proc_table[0].address_i, proc_table[0].size_i);
1710104     //
1710105     // Allocate memory for the data segment if different.
1710106     //
1710107     if (seg_d () != seg_i ())
1710108       {
1710109         mb_alloc (proc_table[0].address_d, proc_table[0].size_d);
1710110       }
1710111     //
1710112     // Allocate memory for the BIOS data area (BDA).
1710113     //
1710114     mb_alloc (0x00000L, 0x500);
1710115     //
1710116     // Allocate memory for the extra BIOS at the
1710117     // bottom of the 640 Kibyte.
1710118     //
1710119     start  = int12 ();
1710120     start *= 1024;
1710121     size   = 0xA0000L - start;
1710122     mb_alloc (start, size);
1710123     //
1710124     // Enable and disable hardware interrupts (IRQ).
1710125     //
1710126     irq_on  (0); // timer.
1710127     irq_on  (1); // enable keyboard
1710128     irq_off (2); //
1710129     irq_off (3); //
1710130     irq_off (4); //
1710131     irq_off (5); //
1710132     irq_on  (6); // floppy (must be on to let int 13 work)!
1710133     irq_off (7); //
1710134     //
1710135     // Interrupts activation.
1710136     //
1710137     sti ();
1710138 }
```

kernel/proc/proc_reference.c

«

Si veda la sezione i159.8.7.

```
1720001 #include <kernel/proc.h>
1720002 //---------------------------------------------------------------------
1720003 proc_t *
1720004 proc_reference (pid_t pid)
1720005 {
1720006     if (pid >= 0 && pid < PROCESS_MAX)
1720007       {
1720008         return (&proc_table[pid]);
1720009       }
1720010     else
1720011       {
1720012         return (NULL);
1720013       }
1720014 }
```

kernel/proc/proc_sch_signals.c

«

Si veda la sezione i159.8.8.

```
1730001 #include <kernel/proc.h>
1730002 //---------------------------------------------------------------------
1730003 void
1730004 proc_sch_signals (void)
1730005 {
1730006     pid_t  pid;
1730007     for (pid = 0; pid < PROCESS_MAX; pid++)
1730008       {
1730009         proc_sig_term (pid, SIGHUP);
1730010         proc_sig_term (pid, SIGINT);
1730011         proc_sig_core (pid, SIGQUIT);
1730012         proc_sig_core (pid, SIGILL);
1730013         proc_sig_core (pid, SIGABRT);
1730014         proc_sig_core (pid, SIGFPE);
1730015         proc_sig_term (pid, SIGKILL);
1730016         proc_sig_core (pid, SIGSEGV);
1730017         proc_sig_term (pid, SIGPIPE);
1730018         proc_sig_term (pid, SIGALRM);
1730019         proc_sig_term (pid, SIGTERM);
1730020         proc_sig_term (pid, SIGUSR1);
1730021         proc_sig_term (pid, SIGUSR2);
1730022         proc_sig_chld (pid, SIGCHLD);
1730023         proc_sig_cont (pid, SIGCONT);
1730024         proc_sig_stop (pid, SIGSTOP);
1730025         proc_sig_stop (pid, SIGTSTP);
1730026         proc_sig_stop (pid, SIGTTIN);
1730027         proc_sig_stop (pid, SIGTTOU);
1730028       }
```

```
1730029 }
```

kernel/proc/proc_sch_terminals.c

«

Si veda la sezione i159.8.9.

```
1740001 #include <kernel/proc.h>
1740002 #include <kernel/k_libc.h>
1740003 //---------------------------------------------------------------------
1740004 void
1740005 proc_sch_terminals (void)
1740006 {
1740007     pid_t  pid;
1740008     int    key;
1740009     tty_t *tty;
1740010     dev_t  device;
1740011     //
1740012     // Try to read a key from console keyboard buffer (only consoles
1740013     // are available).
1740014     //
1740015     key = con_char_ready ();
1740016     if (key == 0)
1740017       {
1740018         //
1740019         // No key is ready on the keyboard buffer: just return.
1740020         //
1740021         return;
1740022       }
1740023     //
1740024     // A key is available. Find the currently active console.
1740025     //
1740026     device = tty_console ((dev_t) 0);
1740027     tty    = tty_reference (device);
1740028     if (tty == NULL)
1740029       {
1740030         k_printf ("kernel alert: console device 0x%04x not found!\n",
1740031                   device);
1740032         //
1740033         // Will send the typed character to the first terminal!
1740034         //
1740035         tty = tty_reference ((dev_t) 0);
1740036       }
1740037     //
1740038     // Defined the active console. Put the character there.
1740039     //
1740040     if (tty->key == 0)
1740041       {
1740042         tty->status = TTY_OK;
1740043       }
1740044     else
1740045       {
1740046         tty->status = TTY_LOST_KEY;
1740047       }
1740048     tty->key = con_char_read ();
1740049     //
1740050     // Verify if it is a control key that must be handled. If so, a
1740051     // signal is sent to all processes with the same control terminal,
1740052     // excluded the kernel (0) and 'init' (1). Such control keys are not
1740053     // passed to the applications.
1740054     //
1740055     // Please note that this a simplified solution, because the signal
1740056     // should reach only the foreground process of the group. For that
1740057     // reason, only che [Ctrl C] is taken into consideration, because
1740058     // processes can ignore the signal 'SIGINT'.
1740059     //
1740060     if (tty->pgrp != 0)
1740061       {
1740062         //
1740063         // There is a process group for that terminal.
1740064         //
1740065         if      (tty->key == 3)         // [Ctrl C] -> SIGINT
1740066           {
1740067             for (pid = 2; pid <  PROCESS_MAX; pid++)
1740068               {
1740069                 if (proc_table[pid].pgrp == tty->pgrp)
1740070                   {
1740071                     k_kill (pid, SIGINT);
1740072                   }
1740073               }
1740074             tty->key    = 0;    // Reset key and status.
1740075             tty->status = TTY_OK;
1740076           }
1740077       }
1740078     //
1740079     // Check for a console switch key combination.
1740080     //
1740081     if      (tty->key == 0x11)          // [Ctrl Q] -> DC1 -> console0.
1740082       {
1740083         tty->key    = 0;                // Reset key and status.
1740084         tty->status = TTY_OK;
1740085         tty_console (DEV_CONSOLE0);     // Switch.
1740086       }
1740087     else if (tty->key == 0x12)          // [Ctrl R] -> DC2 -> console1.
1740088       {
1740089         tty->key    = 0;                // Reset key and status.
1740090         tty->status = TTY_OK;
1740091         tty_console (DEV_CONSOLE1);     // Switch.
1740092       }
1740093     else if (tty->key == 0x13)          // [Ctrl S] -> DC3 -> console2.
1740094       {
1740095         tty->key    = 0;                // Reset key and status.
```



```
1740096         tty->status = TTY_OK;
1740097         tty_console (DEV_CONSOLE2);     // Switch.
1740098       }
1740099     else if (tty->key == 0x14)          // [Ctrl T] -> DC4 -> console3.
1740100       {
1740101         tty->key    = 0;                // Reset key and status.
1740102         tty->status = TTY_OK;
1740103         tty_console (DEV_CONSOLE3);     // Switch.
1740104       }
1740105     //
1740106     // A key was pressed: must wake up all processes waiting for reading
1740107     // a terminal: all processes must be reactivated, because a process
1740108     // can read from the device file, and not just from its own
1740109     // terminal.
1740110     //
1740111     for (pid = 0; pid < PROCESS_MAX; pid++)
1740112       {
1740113         if (   (proc_table[pid].status == PROC_SLEEPING)
1740114             && (proc_table[pid].wakeup_events & WAKEUP_EVENT_TTY))
1740115           {
1740116             //
1740117             // A process waiting for that terminal was found:
1740118             // remove the waiting event and set it ready.
1740119             //
1740120             proc_table[pid].wakeup_events &= ~WAKEUP_EVENT_TTY;
1740121             proc_table[pid].status = PROC_READY;
1740122           }
1740123       }
1740124 }
```

kernel/proc/proc_sch_timers.c

«

Si veda la sezione i159.8.10.

```
1750001 #include <kernel/proc.h>
1750002 #include <kernel/k_libc.h>
1750003 //---------------------------------------------------------------------
1750004 void
1750005 proc_sch_timers (void)
1750006 {
1750007     static unsigned long int previous_time;
1750008            unsigned long int current_time;
1750009            unsigned int      pid;
1750010     current_time = k_time (NULL);
1750011     if (previous_time != current_time)
1750012       {
1750013         for (pid = 0; pid < PROCESS_MAX; pid++)
1750014           {
1750015             if (   (proc_table[pid].wakeup_events & WAKEUP_EVENT_TIMER)
1750016                 && (proc_table[pid].status == PROC_SLEEPING)
1750017                 && (proc_table[pid].wakeup_timer > 0))
1750018               {
1750019                 proc_table[pid].wakeup_timer--;
1750020                 if (proc_table[pid].wakeup_timer == 0)
1750021                   {
1750022                     proc_table[pid].status = PROC_READY;
1750023                   }
1750024               }
1750025           }
1750026       }
1750027     previous_time = current_time;
1750028 }
```

kernel/proc/proc_scheduler.c

«

Si veda la sezione i159.8.11.

```
1760001 #include <kernel/proc.h>
1760002 #include <kernel/k_libc.h>
1760003 #include <stdint.h>
1760004 //---------------------------------------------------------------------
1760005 extern uint16_t _ksp;
1760006 //---------------------------------------------------------------------
1760007 void
1760008 proc_scheduler (uint16_t *sp, segment_t *segment_d)
1760009 {
1760010     //
1760011     // The process is identified from the data and stack segment.
1760012     //
1760013     pid_t prev;
1760014     pid_t next;
1760015     //
1760016     static unsigned long int previous_clock;
1760017            unsigned long int current_clock;
1760018     //
1760019     // Check if current data segments are right.
1760020     //
1760021     if (es () != ds () || ss () != ds ())
1760022       {
1760023         k_printf ("\n");
1760024         k_printf ("Kernel panic: ES, DS, SS are different!\n");
1760025         k_exit (0);
1760026       }
1760027     //
1760028     // Search the data segment inside the process table.
1760029     // Must be done here, because the subsequent call to
1760030     // proc_sch_signals() will remove the segment numbers
1760031     // from a zombie process.
1760032     //
1760033     prev = proc_find (*segment_d);
```

```
1760034     //
1760035     // Take care of sleeping processes: wake up if sleeping time
1760036     // elapsed.
1760037     //
1760038     proc_sch_timers ();
1760039     //
1760040     // Take care of pending signals.
1760041     //
1760042     proc_sch_signals ();
1760043     //
1760044     // Take care input from terminals.
1760045     //
1760046     proc_sch_terminals ();
1760047     //
1760048     // Update the CPU time usage.
1760049     //
1760050     current_clock   = k_clock ();
1760051     proc_table[prev].usage += current_clock - previous_clock;
1760052     previous_clock  = current_clock;
1760053     //
1760054     // Scan for a next process.
1760055     //
1760056     for (next = prev+1; next != prev; next++)
1760057       {
1760058         if (next >= PROCESS_MAX)
1760059           {
1760060             next = -1; // At the next loop, 'next' will be zero.
1760061             continue;
1760062           }
1760063         if      (proc_table[next].status == PROC_EMPTY)
1760064           {
1760065             continue;
1760066           }
1760067         else if (proc_table[next].status == PROC_CREATED)
1760068           {
1760069             continue;
1760070           }
1760071         else if (proc_table[next].status == PROC_READY)
1760072           {
1760073             if (proc_table[prev].status == PROC_RUNNING)
1760074               {
1760075                 proc_table[prev].status = PROC_READY;
1760076               }
1760077             proc_table[prev].sp     = *sp;
1760078             proc_table[next].status = PROC_RUNNING;
1760079             proc_table[next].ret    = 0;
1760080             *segment_d      = proc_table[next].segment_d;
1760081             *sp             = proc_table[next].sp;
1760082             break;
1760083           }
1760084         else if (proc_table[next].status == PROC_RUNNING)
1760085           {
1760086             if (proc_table[prev].status == PROC_RUNNING)
1760087               {
1760088                 k_printf ("Kernel alert: process %i ", prev);
1760089                 k_printf ("and %i \"running\"!\r", next);
1760090                 proc_table[prev].status = PROC_READY;
1760091               }
1760092             proc_table[prev].sp     = *sp;
1760093             proc_table[next].status = PROC_RUNNING;
1760094             proc_table[next].ret    = 0;
1760095             *segment_d      = proc_table[next].segment_d;
1760096             *sp             = proc_table[next].sp;
1760097             break;
1760098           }
1760099         else if (proc_table[next].status == PROC_SLEEPING)
1760100           {
1760101             continue;
1760102           }
1760103         else if (proc_table[next].status == PROC_ZOMBIE)
1760104           {
1760105             continue;
1760106           }
1760107       }
1760108     //
1760109     // Check again if the next process is set to running, otherwise set
1760110     // the kernel to such value!
1760111     //
1760112     next = proc_find (*segment_d);
1760113     if (proc_table[next].status != PROC_RUNNING)
1760114       {
1760115         proc_table[0].status = PROC_RUNNING;
1760116         *segment_d   = proc_table[0].segment_d;
1760117         *sp          = proc_table[0].sp;
1760118       }
1760119     //
1760120     // Save kernel stack pointer.
1760121     //
1760122     _ksp = proc_table[0].sp;
1760123     //
1760124     // At the end, must inform the PIC 1, with message «EOI».
1760125     //
1760126     out_8 (0x20, 0x20);
1760127 }
```


kernel/proc/proc_sig_chld.c

«

Si veda la sezione i159.8.12.

```
1770001 #include <kernel/proc.h>
1770002 //---------------------------------------------------------------------
1770003 void
1770004 proc_sig_chld (pid_t parent, int sig)
1770005 {
1770006     pid_t child;
1770007     //
1770008     // Please note that 'sig' should be SIGCHLD and nothing else.
1770009     // So, the following test, means to verify if the parent process
1770010     // has received a SIGCHLD already.
1770011     //
1770012     if (proc_sig_status (parent, sig))
1770013       {
1770014         if (   (!proc_sig_ignore (parent, sig))
1770015             && (proc_table[parent].status == PROC_SLEEPING)
1770016             && (proc_table[parent].wakeup_events & WAKEUP_EVENT_SIGNAL)
1770017             && (proc_table[parent].wakeup_signal == sig))
1770018           {
1770019             //
1770020             // The signal is not ignored from the parent process;
1770021             // the parent process is sleeping;
1770022             // the parent process is waiting for a signal;
1770023             // the parent process is waiting for current signal.
1770024             // So, just wake it up.
1770025             //
1770026             proc_table[parent].status         = PROC_READY;
1770027             proc_table[parent].wakeup_events ^= WAKEUP_EVENT_SIGNAL;
1770028             proc_table[parent].wakeup_signal  = 0;
1770029           }
1770030         else
1770031           {
1770032             //
1770033             // All other cases, means to remove all dead children.
1770034             //
1770035             for (child = 1; child < PROCESS_MAX; child++)
1770036               {
1770037                 if (   proc_table[child].ppid == parent
1770038                     && proc_table[child].status == PROC_ZOMBIE)
1770039                   {
1770040                     proc_available (child);
1770041                   }
1770042               }
1770043           }
1770044         proc_sig_off (parent, sig);
1770045       }
1770046 }
```

kernel/proc/proc_sig_cont.c

«

Si veda la sezione i159.8.13.

```
1780001 #include <kernel/proc.h>
1780002 //---------------------------------------------------------------------
1780003 void
1780004 proc_sig_cont (pid_t pid, int sig)
1780005 {
1780006     //
1780007     // The value for argument 'sig' should be SIGCONT.
1780008     //
1780009     if (proc_sig_status (pid, sig))
1780010       {
1780011         if (proc_sig_ignore (pid, sig))
1780012           {
1780013             proc_sig_off (pid, sig);
1780014           }
1780015         else
1780016           {
1780017             proc_table[pid].status = PROC_READY;
1780018             proc_sig_off (pid, sig);
1780019           }
1780020       }
1780021 }
```

kernel/proc/proc_sig_core.c

«

Si veda la sezione i159.8.14.

```
1790001 #include <kernel/proc.h>
1790002 //---------------------------------------------------------------------
1790003 void
1790004 proc_sig_core (pid_t pid, int sig)
1790005 {
1790006     addr_t address_i;
1790007     addr_t address_d;
1790008     size_t size_i;
1790009     size_t size_d;
1790010     //
1790011     if (proc_sig_status (pid, sig))
1790012       {
1790013         if (proc_sig_ignore (pid, sig))
1790014           {
1790015             proc_sig_off (pid, sig);
1790016           }
1790017         else
1790018           {
1790019             //
```

```
1790020             // Save process addresses and sizes (might be useful if
1790021             // we want to try to exit the process before core dump.
1790022             //
1790023             address_i   = proc_table[pid].address_i;
1790024             address_d   = proc_table[pid].address_d;
1790025             size_i      = proc_table[pid].size_i;
1790026             size_d      = proc_table[pid].size_d;
1790027             //
1790028             // Core dump: the process who formally writes the file
1790029             // is the terminating one.
1790030             //
1790031             if (address_d == address_i)
1790032               {
1790033                 proc_dump_memory (pid, address_i, size_i, "core");
1790034               }
1790035             else
1790036               {
1790037                 proc_dump_memory (pid, address_i, size_i, "core.i");
1790038                 proc_dump_memory (pid, address_d, size_d, "core.d");
1790039               }
1790040             //
1790041             // The signal, translated to negative, is returned (but
1790042             // the effective value received by the application will
1790043             // be cutted, leaving only the low 8 bit).
1790044             //
1790045             proc_sys_exit (pid, -sig);
1790046           }
1790047       }
1790048 }
```

kernel/proc/proc_sig_ignore.c

Si veda la sezione i159.8.15.

```
1800001 #include <kernel/proc.h>
1800002 //-------------------------------------------------------------------------
1800003 int
1800004 proc_sig_ignore (pid_t pid, int sig)
1800005 {
1800006     unsigned long int flag = 1L << (sig - 1);
1800007     if (proc_table[pid].sig_ignore & flag)
1800008       {
1800009         return (1);
1800010       }
1800011     else
1800012       {
1800013         return (0);
1800014       }
1800015 }
```

kernel/proc/proc_sig_off.c

Si veda la sezione i159.8.16.

```
1810001 #include <kernel/proc.h>
1810002 //-------------------------------------------------------------------------
1810003 void
1810004 proc_sig_off (pid_t pid, int sig)
1810005 {
1810006     unsigned long int flag = 1L << (sig - 1);
1810007     proc_table[pid].sig_status ^= flag;
1810008 }
```

kernel/proc/proc_sig_on.c

Si veda la sezione i159.8.16.

```
1820001 #include <kernel/proc.h>
1820002 //-------------------------------------------------------------------------
1820003 void
1820004 proc_sig_on (pid_t pid, int sig)
1820005 {
1820006     unsigned long int flag = 1L << (sig - 1);
1820007     proc_table[pid].sig_status |= flag;
1820008 }
```

kernel/proc/proc_sig_status.c

Si veda la sezione i159.8.17.

```
1830001 #include <kernel/proc.h>
1830002 //-------------------------------------------------------------------------
1830003 int
1830004 proc_sig_status (pid_t pid, int sig)
1830005 {
1830006     unsigned long int flag = 1L << (sig - 1);
1830007     if (proc_table[pid].sig_status & flag)
1830008       {
1830009         return (1);
1830010       }
1830011     else
1830012       {
1830013         return (0);
1830014       }
1830015 }
```


kernel/proc/proc_sig_stop.c

Si veda la sezione i159.8.18.

```
1840001 #include <kernel/proc.h>
1840002 //-------------------------------------------------------------------------
1840003 void
1840004 proc_sig_stop (pid_t pid, int sig)
1840005 {
1840006     if (proc_sig_status (pid, sig))
1840007       {
1840008         if (proc_sig_ignore (pid, sig) && !(sig == SIGSTOP))
1840009           {
1840010             proc_sig_off (pid, sig);
1840011           }
1840012         else
1840013           {
1840014             proc_table[pid].status = PROC_SLEEPING;
1840015             proc_table[pid].ret    = -sig;
1840016             proc_sig_off (pid, sig);
1840017           }
1840018       }
1840019 }
```

kernel/proc/proc_sig_term.c

Si veda la sezione i159.8.19.

```
1850001 #include <kernel/proc.h>
1850002 //-------------------------------------------------------------------------
1850003 void
1850004 proc_sig_term (pid_t pid, int sig)
1850005 {
1850006     if (proc_sig_status (pid, sig))
1850007       {
1850008         if (proc_sig_ignore (pid, sig) && !(sig == SIGKILL))
1850009           {
1850010             proc_sig_off (pid, sig);
1850011           }
1850012         else
1850013           {
1850014             //
1850015             // The signal, translated to negative, is returned (but
1850016             // the effective value received by the application will
1850017             // be cutted, leaving only the low 8 bit).
1850018             //
1850019             proc_sys_exit (pid, -sig);
1850020           }
1850021       }
1850022 }
```

kernel/proc/proc_sys_exec.c

Si veda la sezione i159.8.20.

```
1860001 #include <kernel/proc.h>
1860002 #include <errno.h>
1860003 #include <fcntl.h>
1860004 //-------------------------------------------------------------------------
1860005 int
1860006 proc_sys_exec (uint16_t *sp, segment_t *segment_d, pid_t pid,
1860007                const char *path,
1860008                unsigned int argc, char *arg_data,
1860009                unsigned int envc, char *env_data)
1860010 {
1860011   unsigned int      i;
1860012   unsigned int      j;
1860013   char             *arg;
1860014   char             *env;
1860015   char             *envp[ARG_MAX/16];
1860016   char             *argv[ARG_MAX/16];
1860017   size_t            size;
1860018   size_t            arg_data_size;
1860019   size_t            env_data_size;
1860020   unsigned int      p_off;
1860021   inode_t          *inode;
1860022   ssize_t           size_read;
1860023   header_t          header;
1860024   unsigned long int s;        // Help calculating process sizes.
1860025   uint16_t          new_sp;
1860026   uint16_t          envp_address;
1860027   uint16_t          argv_address;
1860028   size_t            process_size_i;
1860029   size_t            process_size_d;
1860030   char              buffer[MEM_BLOCK_SIZE];
1860031   uint16_t          stack_element;
1860032   off_t             inode_start;
1860033   addr_t            memory_start;
1860034   addr_t            previous_address_i;
1860035   segment_t         previous_segment_i;
1860036   size_t            previous_size_i;
1860037   addr_t            previous_address_d;
1860038   segment_t         previous_segment_d;
1860039   size_t            previous_size_d;
1860040   int               status;
1860041   memory_t          allocated_i;
1860042   memory_t          allocated_d;
1860043   pid_t             extra;
1860044   int               proc_count;
1860045   file_t           *file;
```

```
  int               fdn;
  dev_t             device;
  int               eof;
  //
  // Check for limits.
  //
  if (argc > (ARG_MAX/16) || envc > (ARG_MAX/16))
    {
      errset (ENOMEM);
      return (-1);
    }
  //
  // Scan arguments to calculate the full size and the relative
  // pointers. The final size is rounded to 2, for the stack.
  //
  arg = arg_data;
  for (i = 0, j = 0; i < argc; i++)
    {
      argv[i] = (char *) j;   // Relative pointer inside
                              // the 'arg_data'.
      size = strlen (arg);
      arg += size + 1;
      j += size + 1;
    }
  arg_data_size = j;
  if (arg_data_size % 2)
    {
      arg_data_size++;
    }
  //
  // Scan environment variables to calculate the full size and the
  // relative pointers. The final size is rounded to 2, for the stack.
  //
  env = env_data;
  for (i = 0, j = 0; i < envc; i++)
    {
      envp[i] = (char *) j;   // Relative pointer inside
                              // the 'env_data'.
      size = strlen (env);
      env += size + 1;
      j += size + 1;
    }
  env_data_size = j;
  if (env_data_size % 2)
    {
      env_data_size++;
    }
  //
  // Read the inode related to the executable file name.
  // Function path_inode() includes the inode get procedure.
  //
  inode = path_inode (pid, path);
  if (inode == NULL)
    {
      errset (ENOENT);        // No such file or directory.
      return (-1);
    }
  //
  // Check for permissions.
  //
  status = inode_check (inode, S_IFREG, 5, proc_table[pid].euid);
  if (status != 0)
    {
      //
      // File is not of a valid type or permission are not
      // sufficient: release the executable file inode
      // and return with an error.
      //
      inode_put (inode);
      errset (EACCES);        // Permission denied.
      return (-1);
    }
  //
  // Read the header from the executable file.
  //
  size_read = inode_file_read (inode, (off_t) 0, &header,
                               (sizeof header), &eof);
  if (size_read != (sizeof header))
    {
      //
      // The file is shorter than the executable header, so, it isn't
      // an executable: release the file inode and return with an
      // error.
      //
      inode_put (inode);
      errset (ENOEXEC);
      return (-1);
    }
  if (header.magic0 != MAGIC_OS16 || header.magic1 != MAGIC_OS16_APPL)
    {
      //
      // The header does not have the expected magic numbers, so,
      // it isn't a valid executable: release the file inode and
      // return with an error.
      //
      inode_put (inode);
      errset (ENOEXEC);
      return (-1);    // This is not a valid executable!
    }
  //
  // Calculate data size.
```



```
  //
  s  = header.ebss;
  if (header.ssize == 0)
    {
      s += 0x10000L;  // Zero means max size.
    }
  else
    {
      s += header.ssize;
    }
  if (s > 0xFFFF)
    {
      process_size_d = 0x0000;  // 0x0000 means the maximum size:
                                // 0x10000.
      new_sp         = 0x0000;  // 0x0000 is like 0x10000 and the first
                                // push moves SP to 0xFFFE.
    }
  else
    {
      process_size_d = s;
      new_sp         = process_size_d;
      if (new_sp % 2)
        {
          new_sp--;       // The stack pointer should be even.
        }
    }
  //
  // Calculate code size.
  //
  if (header.segoff == 0)
    {
      process_size_i = process_size_d;
    }
  else
    {
      process_size_i = header.segoff * 16;
    }
  //
  // Allocate memory: code and data segments.
  //
  status = mb_alloc_size (process_size_i, &allocated_i);
  if (status < 0)
    {
      //
      // The program instructions (code segment) cannot be loaded
      // into memory: release the executable file inode and return
      // with an error.
      //
      inode_put (inode);
      errset (ENOMEM);        // Not enough space.
      return ((pid_t) -1);
    }
  if (header.segoff == 0)
    {
      //
      // Code and data segments are the same: no need
      // to allocate more memory for the data segment.
      //
      allocated_d.address = allocated_i.address;
      allocated_d.segment = allocated_i.segment;
      allocated_d.size    = allocated_i.size;
    }
  else
    {
      //
      // Code and data segments are different: the data
      // segment memory is allocated.
      //
      status = mb_alloc_size (process_size_d, &allocated_d);
      if (status < 0)
        {
          //
          // The separated program data (data segment) cannot be loaded
          // into memory: free the already allocated memory for the
          // program instructions, release the executable file inode
          // and return with an error.
          //
          mb_free (allocated_i.address, allocated_i.size);
          inode_put (inode);
          errset (ENOMEM);        // Not enough space.
          return ((pid_t) -1);
        }
    }
  //
  // Load executable in memory.
  //
  if (header.segoff == 0)
    {
      //
      // Code and data share the same segment.
      //
      for (eof = 0, memory_start = allocated_i.address,
           inode_start = 0, size_read = 0;
           inode_start < inode->size && !eof;
           inode_start += size_read)
        {
          memory_start += size_read;
          //
          // Read a block of memory.
          //
          size_read = inode_file_read (inode, inode_start,
```

```
                                    buffer, MEM_BLOCK_SIZE, &eof);
          if (size_read < 0)
            {
              //
              // Free memory and inode.
              //
              mb_free (allocated_i.address, allocated_i.size);
              inode_put (inode);
              errset (EIO);
              return (-1);
            }
          //
          // Copy inside the right position to be executed.
          //
          dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE, memory_start, buffer,
                  (size_t) size_read, NULL);
        }
    }
  else
    {
      //
      // Code and data with different segments.
      //
      for (eof = 0, memory_start = allocated_i.address,
           inode_start = 0, size_read = 0;
           inode_start < process_size_i && !eof;
           inode_start += size_read)
        {
          memory_start += size_read;
          //
          // Read a block of memory
          //
          size_read = inode_file_read (inode, inode_start,
                                       buffer, MEM_BLOCK_SIZE, &eof);
          if (size_read < 0)
            {
              //
              // Free memory and inode.
              //
              mb_free (allocated_i.address, allocated_i.size);
              mb_free (allocated_d.address, allocated_d.size);
              inode_put (inode);
              errset (EIO);
              return (-1);
            }
          //
          // Copy inside the right position to be executed.
          //
          dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE, memory_start, buffer,
                  (size_t) size_read, NULL);
        }
      for (eof = 0, memory_start = allocated_d.address,
           inode_start = (header.segoff * 16), size_read = 0;
           inode_start < inode->size && !eof;
           inode_start += size_read)
        {
          memory_start += size_read;
          //
          // Read a block of memory
          //
          size_read = inode_file_read (inode, inode_start,
                                       buffer, MEM_BLOCK_SIZE, &eof);
          if (size_read < 0)
            {
              //
              // Free memory and inode.
              //
              mb_free (allocated_i.address, allocated_i.size);
              mb_free (allocated_d.address, allocated_d.size);
              inode_put (inode);
              errset (EIO);
              return (-1);
            }
          dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE, memory_start, buffer,
                  (size_t) size_read, NULL);
        }
    }
  //
  // The executable file was successfully loaded in memory:
  // release the executable file inode.
  //
  inode_put (inode);
  //
  // Put environment data inside the stack.
  //
  new_sp -= env_data_size; //------------------------------ environment
  mem_copy (address (seg_d (), (unsigned int) env_data),
            (allocated_d.address + new_sp), env_data_size);
  //
  // Put arguments data inside the stack.
  //
  new_sp -= arg_data_size; //-------------------------------- arguments
  mem_copy (address (seg_d (), (unsigned int) arg_data),
            (allocated_d.address + new_sp), arg_data_size);
  //
  // Put envp[] inside the stack, updating all the pointers.
  //
  new_sp -= 2; //------------------------------------------------ NULL
  stack_element = NULL;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
```



```
          &stack_element, (sizeof stack_element), NULL);
  //
  p_off  = new_sp;            //
  p_off += 2;                 // Calculate memory pointers from
  p_off += arg_data_size;     // original relative pointers,
  for (i = 0; i < envc; i++)  // inside the environment array
    {                         // of pointers.
      envp[i] += p_off;       //
    }                         //
  //
  new_sp -= (envc * (sizeof (char *))); //---------------------- *envp[]
  mem_copy (address (seg_d (), (unsigned int) envp),
            (allocated_d.address + new_sp),
            (envc * (sizeof (char *))));
  //
  // Save the envp[] location, needed in the following.
  //
  envp_address = new_sp;
  //
  // Put argv[] inside the stack, updating all the pointers.
  //
  new_sp -= 2; //------------------------------------------------ NULL
  stack_element = NULL;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  //
  p_off  = new_sp;                      //
  p_off += 2 ;                          // Calculate memory pointers
  p_off += (envc * (sizeof (char *)));  // from original relative
  p_off += 2;                           // pointers, inside the
  for (i = 0; i < argc; i++)            // arguments array of
    {                                   // pointers.
      argv[i] += p_off;                 //
    }                                   //
  //
  new_sp -= (argc * (sizeof (char *))); //---------------------- *argv[]
  mem_copy (address (seg_d (), (unsigned int) argv),
            (allocated_d.address + new_sp),
            (argc * (sizeof (char *))));
  //
  // Save the argv[] location, needed in the following.
  //
  argv_address = new_sp;
  //
  // Put the pointer to the array envp[].
  //
  new_sp -= 2; //------------------------------------------------ argc
  stack_element = envp_address;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  //
  // Put the pointer to the array argv[].
  //
  new_sp -= 2; //------------------------------------------------ argc
  stack_element = argv_address;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  //
  // Put argc inside the stack.
  //
  new_sp -= 2; //------------------------------------------------ argc
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &argc, (sizeof argc), NULL);
  //
  // Set the rest of the stack.
  //
  new_sp -= 2; //----------------------------------------------- FLAGS
  stack_element = 0x0200;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  new_sp -= 2; //------------------------------------------------- CS
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &allocated_i.segment, (sizeof allocated_i.segment), NULL);
  new_sp -= 2; //------------------------------------------------ IP
  stack_element = 0;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  new_sp -= 2; //------------------------------------------------ ES
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &allocated_d.segment, (sizeof allocated_d.segment), NULL);
  new_sp -= 2; //------------------------------------------------- DS
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &allocated_d.segment, (sizeof allocated_d.segment), NULL);
  new_sp -= 2; //------------------------------------------------ DI
  stack_element = 0;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
          &stack_element, (sizeof stack_element), NULL);
  new_sp -= 2; //------------------------------------------------ SI
  stack_element = 0;
  dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
          (allocated_d.address + new_sp),
```

```
1860450           &stack_element, (sizeof stack_element), NULL);
1860451   new_sp -= 2; //----------------------------------------------------- BP
1860452   stack_element = 0;
1860453   dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1860454           (allocated_d.address + new_sp),
1860455           &stack_element, (sizeof stack_element), NULL);
1860456   new_sp -= 2; //----------------------------------------------------- BX
1860457   stack_element = 0;
1860458   dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1860459           (allocated_d.address + new_sp),
1860460           &stack_element, (sizeof stack_element), NULL);
1860461   new_sp -= 2; //----------------------------------------------------- DX
1860462   stack_element = 0;
1860463   dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1860464           (allocated_d.address + new_sp),
1860465           &stack_element, (sizeof stack_element), NULL);
1860466   new_sp -= 2; //----------------------------------------------------- CX
1860467   stack_element = 0;
1860468   dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1860469           (allocated_d.address + new_sp),
1860470           &stack_element, (sizeof stack_element), NULL);
1860471   new_sp -= 2; //----------------------------------------------------- AX
1860472   stack_element = 0;
1860473   dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1860474           (allocated_d.address + new_sp),
1860475           &stack_element, (sizeof stack_element), NULL);
1860476   //
1860477   // Close process file descriptors, if the 'FD_CLOEXEC' flag
1860478   // is present.
1860479   //
1860480   for (fdn = 0; fdn < OPEN_MAX; fdn++)
1860481     {
1860482       if (proc_table[pid].fd[0].file != NULL)
1860483         {
1860484           if (proc_table[pid].fd[0].fd_flags & FD_CLOEXEC)
1860485             {
1860486               fd_close (pid, fdn);
1860487             }
1860488         }
1860489     }
1860490   //
1860491   // Select device for standard I/O, if a standard I/O stream must be
1860492   // opened.
1860493   //
1860494   if (proc_table[pid].device_tty != 0)
1860495     {
1860496       device = proc_table[pid].device_tty;
1860497     }
1860498   else
1860499     {
1860500       device = DEV_TTY;
1860501     }
1860502   //
1860503   // Prepare missing standard file descriptors. The function
1860504   // 'file_stdio_dev_make()' arranges the value for 'errno' if
1860505   // necessary. If a standard file descriptor cannot be allocated,
1860506   // the program is left without it.
1860507   //
1860508   if (proc_table[pid].fd[0].file == NULL)
1860509     {
1860510       file = file_stdio_dev_make (device, S_IFCHR, O_RDONLY);
1860511       if (file != NULL)                                    // stdin
1860512         {
1860513           proc_table[pid].fd[0].fl_flags     = O_RDONLY;
1860514           proc_table[pid].fd[0].fd_flags     = 0;
1860515           proc_table[pid].fd[0].file         = file;
1860516           proc_table[pid].fd[0].file->offset = 0;
1860517         }
1860518     }
1860519   if (proc_table[pid].fd[1].file == NULL)
1860520     {
1860521       file = file_stdio_dev_make (device, S_IFCHR, O_WRONLY);
1860522       if (file != NULL)                                   // stdout
1860523         {
1860524           proc_table[pid].fd[1].fl_flags     = O_WRONLY;
1860525           proc_table[pid].fd[1].fd_flags     = 0;
1860526           proc_table[pid].fd[1].file         = file;
1860527           proc_table[pid].fd[1].file->offset = 0;
1860528         }
1860529     }
1860530   if (proc_table[pid].fd[2].file == NULL)
1860531     {
1860532       file = file_stdio_dev_make (device, S_IFCHR, O_WRONLY);
1860533       if (file != NULL)                                   // stderr
1860534         {
1860535           proc_table[pid].fd[2].fl_flags     = O_WRONLY;
1860536           proc_table[pid].fd[2].fd_flags     = 0;
1860537           proc_table[pid].fd[2].file         = file;
1860538           proc_table[pid].fd[2].file->offset = 0;
1860539         }
1860540     }
1860541   //
1860542   // Prepare to switch
1860543   //
1860544   previous_address_i = proc_table[pid].address_i;
1860545   previous_segment_i = proc_table[pid].segment_i;
1860546   previous_size_i    = proc_table[pid].size_i;
1860547   previous_address_d = proc_table[pid].address_d;
1860548   previous_segment_d = proc_table[pid].segment_d;
1860549   previous_size_d    = proc_table[pid].size_d;
1860550   //
```



```
1860551   proc_table[pid].address_i  = allocated_i.address;
1860552   proc_table[pid].segment_i  = allocated_i.segment;
1860553   proc_table[pid].size_i     = allocated_i.size;
1860554   proc_table[pid].address_d  = allocated_d.address;
1860555   proc_table[pid].segment_d  = allocated_d.segment;
1860556   proc_table[pid].size_d     = allocated_d.size;
1860557   proc_table[pid].sp         = new_sp;
1860558   strncpy (proc_table[pid].name, path, PATH_MAX);
1860559   //
1860560   // Ensure to have a terminated string.
1860561   //
1860562   proc_table[pid].name[PATH_MAX-1] = 0;
1860563   //
1860564   // Free data segment memory.
1860565   //
1860566   mb_free (previous_address_d, previous_size_d);
1860567   //
1860568   // Free code segment memory if it is
1860569   // different from the data segment.
1860570   //
1860571   if (previous_segment_i != previous_segment_d)
1860572     {
1860573       //
1860574       // Must verify if no other process is
1860575       // using the same memory.
1860576       //
1860577       for (proc_count = 0, extra = 0; extra < PROCESS_MAX; extra++)
1860578         {
1860579           if (proc_table[extra].status == PROC_EMPTY   ||
1860580               proc_table[extra].status == PROC_ZOMBIE)
1860581             {
1860582               continue;
1860583             }
1860584           if (previous_segment_i == proc_table[extra].segment_i)
1860585             {
1860586               proc_count++;
1860587             }
1860588         }
1860589       if (proc_count == 0)
1860590         {
1860591           //
1860592           // The code segment can be released, because no other
1860593           // process is using it.
1860594           //
1860595           mb_free (previous_address_i, previous_size_i);
1860596         }
1860597     }
1860598   //
1860599   // Change the segment and the stack pointer, from the interrupt.
1860600   //
1860601   *segment_d = proc_table[pid].segment_d;
1860602   *sp        = proc_table[pid].sp;
1860603   //
1860604   return (0);
1860605 }
```

kernel/proc/proc_sys_exit.c

«

Si veda la sezione i159.8.21.

```
1870001 #include <kernel/proc.h>
1870002 #include <kernel/k_libc.h>
1870003 //---------------------------------------------------------------------
1870004 void
1870005 proc_sys_exit (pid_t pid, int status)
1870006 {
1870007   pid_t  child;
1870008   pid_t  parent = proc_table[pid].ppid;
1870009   pid_t  extra;
1870010   int    proc_count;
1870011   int    sigchld = 0;
1870012   int    fdn;
1870013   tty_t *tty;
1870014   //
1870015   proc_table[pid].status     = PROC_ZOMBIE;
1870016   proc_table[pid].ret        = status;
1870017   proc_table[pid].sig_status =  0;
1870018   proc_table[pid].sig_ignore =  0;
1870019   //
1870020   // Close files.
1870021   //
1870022   for (fdn = 0; fdn < OPEN_MAX; fdn++)
1870023     {
1870024       fd_close (pid, fdn);
1870025     }
1870026   //
1870027   // Close current directory.
1870028   //
1870029   inode_put (proc_table[pid].inode_cwd);
1870030   //
1870031   // Close the controlling terminal, if it is a process leader with
1870032   // such a terminal.
1870033   //
1870034   if (proc_table[pid].pgrp == pid && proc_table[pid].device_tty != 0)
1870035     {
1870036       tty = tty_reference (proc_table[pid].device_tty);
1870037       //
1870038       // Verify.
1870039       //
1870040       if (tty == NULL)
1870041         {
```

```
1870042           //
1870043           // Show a kernel message.
1870044           //
1870045           k_printf ("kernel alert: cannot find the terminal item "
1870046                     "for device 0x%04x!\n",
1870047                     (int) proc_table[pid].device_tty);
1870048         }
1870049       else if (tty->pgrp != pid)
1870050         {
1870051           //
1870052           // Show a kernel message.
1870053           //
1870054           k_printf ("kernel alert: terminal device 0x%04x should "
1870055                     "be associated to the process group %i, but it "
1870056                     "is instead related to process group %i!\n",
1870057                     (int) proc_table[pid].device_tty, (int) pid,
1870058                     (int) tty->pgrp);
1870059         }
1870060       else
1870061         {
1870062           tty->pgrp = 0;
1870063         }
1870064     }
1870065   //
1870066   // Free data segment memory.
1870067   //
1870068   mb_free (proc_table[pid].address_d, proc_table[pid].size_d);
1870069   //
1870070   // Free code segment memory if it is
1870071   // different from the data segment.
1870072   //
1870073   if (proc_table[pid].segment_i != proc_table[pid].segment_d)
1870074     {
1870075       //
1870076       // Must verify if no other process is using the same memory.
1870077       // The proc_count variable is incremented for processes
1870078       // active, ready or sleeping: the current process is already
1870079       // set as zombie, and is not counted.
1870080       //
1870081       for (proc_count = 0, extra = 0; extra < PROCESS_MAX; extra++)
1870082         {
1870083           if (proc_table[extra].status == PROC_EMPTY   ||
1870084               proc_table[extra].status == PROC_ZOMBIE)
1870085             {
1870086               continue;
1870087             }
1870088           if (proc_table[pid].segment_i == proc_table[extra].segment_i)
1870089             {
1870090               proc_count++;
1870091             }
1870092         }
1870093       if (proc_count == 0)
1870094         {
1870095           //
1870096           // The code segment can be released, because no other
1870097           // process, except the current one (to be closed),
1870098           // is using it.
1870099           //
1870100           mb_free (proc_table[pid].address_i, proc_table[pid].size_i);
1870101         }
1870102     }
1870103   //
1870104   // Abandon children to 'init' ((pid_t) 1).
1870105   //
1870106   for (child = 1; child < PROCESS_MAX; child++)
1870107     {
1870108       if (   proc_table[child].status != PROC_EMPTY
1870109           && proc_table[child].ppid == pid)
1870110         {
1870111           proc_table[child].ppid = 1; // Son of 'init'.
1870112           if (proc_table[child].status == PROC_ZOMBIE)
1870113             {
1870114               sigchld = 1;    // Must send a SIGCHLD to 'init'.
1870115             }
1870116         }
1870117     }
1870118   //
1870119   // SIGCHLD to 'init'.
1870120   //
1870121   if (   sigchld
1870122       && pid != 1
1870123       && proc_table[1].status != PROC_EMPTY
1870124       && proc_table[1].status != PROC_ZOMBIE)
1870125     {
1870126       proc_sig_on ((pid_t) 1, SIGCHLD);
1870127     }
1870128   //
1870129   // Announce to the parent the death of its child.
1870130   //
1870131   if (   pid != parent
1870132       && proc_table[parent].status != PROC_EMPTY)
1870133     {
1870134       proc_sig_on (parent, SIGCHLD);
1870135     }
1870136 }
```


Si veda la sezione i159.8.22.

```
1880001 #include <kernel/proc.h>
1880002 #include <errno.h>
1880003 //---------------------------------------------------------------------
1880004 pid_t
1880005 proc_sys_fork (pid_t ppid, uint16_t sp)
1880006 {
1880007     pid_t     pid;
1880008     pid_t     zombie;
1880009     memory_t  allocated_i;
1880010     memory_t  allocated_d;
1880011     int       status;
1880012     int       fdn;
1880013     //
1880014     // Find a free PID.
1880015     //
1880016     for (pid = 1; pid < PROCESS_MAX; pid++)
1880017       {
1880018         if (proc_table[pid].status == PROC_EMPTY)
1880019           {
1880020             break;
1880021           }
1880022       }
1880023     if (pid >= PROCESS_MAX)
1880024       {
1880025         //
1880026         // There is no free pid.
1880027         //
1880028         errset (ENOMEM);        // Not enough space.
1880029         return (-1);
1880030       }
1880031     //
1880032     // Before allocating a new process, must check if there are some
1880033     // zombie slots, still with original segment data: should reset
1880034     // it now!
1880035     //
1880036     for (zombie = 1; zombie < PROCESS_MAX; zombie++)
1880037       {
1880038         if (   proc_table[zombie].status == PROC_ZOMBIE
1880039             && proc_table[zombie].segment_d != -1)
1880040           {
1880041             proc_table[zombie].segment_i = -1;    // Reset
1880042             proc_table[zombie].address_i = -1L;   // memory
1880043             proc_table[zombie].size_i    =  0;    // allocation
1880044             proc_table[zombie].segment_d = -1;    // data
1880045             proc_table[zombie].address_d = -1L;   // to
1880046             proc_table[zombie].size_d    =  0;    // impossible
1880047             proc_table[zombie].sp        =  0;    // values.
1880048           }
1880049       }
1880050     //
1880051     // Allocate memory: code and data segments.
1880052     //
1880053     if (proc_table[ppid].segment_i == proc_table[ppid].segment_d)
1880054       {
1880055         //
1880056         // Code segment and Data segment are the same
1880057         // (same I&D).
1880058         //
1880059         status = mb_alloc_size (proc_table[ppid].size_i, &allocated_i);
1880060         if (status < 0)
1880061           {
1880062             errset (ENOMEM);        // Not enough space.
1880063             return ((pid_t) -1);
1880064           }
1880065         allocated_d.address = allocated_i.address;
1880066         allocated_d.segment = allocated_i.segment;
1880067         allocated_d.size    = allocated_i.size;
1880068       }
1880069     else
1880070       {
1880071         //
1880072         // Code segment and Data segment are different
1880073         // (different I&D).
1880074         // Only the data segment is allocated.
1880075         //
1880076         status = mb_alloc_size (proc_table[ppid].size_d, &allocated_d);
1880077         if (status < 0)
1880078           {
1880079             errset (ENOMEM);        // Not enough space.
1880080             return ((pid_t) -1);
1880081           }
1880082         //
1880083         // Code segment is the same from the parent process.
1880084         //
1880085         allocated_i.address = proc_table[ppid].address_i;
1880086         allocated_i.segment = proc_table[ppid].segment_i;
1880087         allocated_i.size    = proc_table[ppid].size_i;
1880088       }
1880089     //
1880090     // Copy the process in memory.
1880091     //
1880092     if (proc_table[ppid].segment_i == proc_table[ppid].segment_d)
1880093       {
1880094         //
1880095         // Code segment and data segment are the same:
1880096         // must copy all.
1880097         //
1880098         // Copy the code segment: if the size is zero,
```

```
1880099         // it means 0x10000 bytes (65536 bytes).
1880100         //
1880101         if (proc_table[ppid].size_i == 0)
1880102           {
1880103             //
1880104             // Copy 0x10000 bytes with two steps.
1880105             //
1880106             mem_copy (proc_table[ppid].address_i,
1880107                       allocated_i.address, 0x8000);
1880108             mem_copy ((proc_table[ppid].address_i + 0x8000),
1880109                       (allocated_i.address + 0x8000), 0x8000);
1880110           }
1880111         else
1880112           {
1880113             //
1880114             // Normal copy.
1880115             //
1880116             mem_copy (proc_table[ppid].address_i, allocated_i.address,
1880117                       proc_table[ppid].size_i);
1880118           }
1880119       }
1880120     else
1880121       {
1880122         //
1880123         // Code segment and data segment are different:
1880124         // copy only the data segment.
1880125         //
1880126         // Copy the data segment in memory: if the size is zero,
1880127         // it means 0x10000 bytes (65536 bytes).
1880128         //
1880129         if (proc_table[ppid].size_d == 0)
1880130           {
1880131             //
1880132             // Copy 0x10000 bytes with two steps.
1880133             //
1880134             mem_copy (proc_table[ppid].address_d,
1880135                       allocated_d.address, 0x8000);
1880136             mem_copy ((proc_table[ppid].address_d + 0x8000),
1880137                       (allocated_d.address + 0x8000), 0x8000);
1880138           }
1880139         else
1880140           {
1880141             //
1880142             // Normal copy.
1880143             //
1880144             mem_copy (proc_table[ppid].address_d, allocated_d.address,
1880145                       proc_table[ppid].size_d);
1880146           }
1880147       }
1880148     //
1880149     // Allocate the new PID.
1880150     //
1880151     proc_table[pid].ppid          = ppid;
1880152     proc_table[pid].pgrp          = proc_table[ppid].pgrp;
1880153     proc_table[pid].uid           = proc_table[ppid].uid;
1880154     proc_table[pid].euid          = proc_table[ppid].euid;
1880155     proc_table[pid].suid          = proc_table[ppid].suid;
1880156     proc_table[pid].device_tty    = proc_table[ppid].device_tty;
1880157     proc_table[pid].sig_status    = 0;
1880158     proc_table[pid].sig_ignore    = 0;
1880159     proc_table[pid].usage         = 0;
1880160     proc_table[pid].status        = PROC_CREATED;
1880161     proc_table[pid].wakeup_events = 0;
1880162     proc_table[pid].wakeup_signal = 0;
1880163     proc_table[pid].wakeup_timer  = 0;
1880164     proc_table[pid].segment_i     = allocated_i.segment;
1880165     proc_table[pid].address_i     = allocated_i.address;
1880166     proc_table[pid].size_i        = proc_table[ppid].size_i;
1880167     proc_table[pid].segment_d     = allocated_d.segment;
1880168     proc_table[pid].address_d     = allocated_d.address;
1880169     proc_table[pid].size_d        = proc_table[ppid].size_d;
1880170     proc_table[pid].sp            = sp;
1880171     proc_table[pid].ret           = 0;
1880172     proc_table[pid].inode_cwd     = proc_table[ppid].inode_cwd;
1880173     proc_table[pid].umask         = proc_table[ppid].umask;
1880174     strncpy (proc_table[pid].name,
1880175              proc_table[ppid].name, PATH_MAX);
1880176     strncpy (proc_table[pid].path_cwd,
1880177              proc_table[ppid].path_cwd, PATH_MAX);
1880178     //
1880179     // Increase inode references for the working directory.
1880180     //
1880181     proc_table[pid].inode_cwd->references++;
1880182     //
1880183     // Duplicate valid file descriptors.
1880184     //
1880185     for (fdn = 0; fdn < OPEN_MAX; fdn++)
1880186       {
1880187         if (   proc_table[ppid].fd[fdn].file != NULL
1880188             && proc_table[ppid].fd[fdn].file->inode != NULL)
1880189           {
1880190             //
1880191             // Copy to the forked process.
1880192             //
1880193             proc_table[pid].fd[fdn].fl_flags
1880194                 = proc_table[ppid].fd[fdn].fl_flags;
1880195             proc_table[pid].fd[fdn].fd_flags
1880196                 = proc_table[ppid].fd[fdn].fd_flags;
1880197             proc_table[pid].fd[fdn].file
1880198                 = proc_table[ppid].fd[fdn].file;
1880199             //
```



```
1880200             // Increment file reference.
1880201             //
1880202             proc_table[ppid].fd[fdn].file->references++;
1880203           }
1880204       }
1880205     //
1880206     // Change segment values inside the stack: DS==ES; CS.
1880207     //
1880208     dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1880209             (allocated_d.address + proc_table[pid].sp + 14),
1880210             &allocated_d.segment, (sizeof allocated_d.segment), NULL);
1880211     dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1880212             (allocated_d.address + proc_table[pid].sp + 16),
1880213             &allocated_d.segment, (sizeof allocated_d.segment), NULL);
1880214     dev_io ((pid_t) 0, DEV_MEM, DEV_WRITE,
1880215             (allocated_d.address + proc_table[pid].sp + 20),
1880216             &allocated_i.segment, (sizeof allocated_i.segment), NULL);
1880217     //
1880218     // Set it ready.
1880219     //
1880220     proc_table[pid].status  = PROC_READY;
1880221     //
1880222     // Return the new PID.
1880223     //
1880224     return (pid);
1880225 }
```

kernel/proc/proc_sys_kill.c

Si veda la sezione i159.8.23.

```
1890001 #include <kernel/proc.h>
1890002 #include <errno.h>
1890003 //---------------------------------------------------------------------
1890004 int
1890005 proc_sys_kill (pid_t pid_killer, pid_t pid_target, int sig)
1890006 {
1890007     uid_t euid = proc_table[pid_killer].euid;
1890008     uid_t uid  = proc_table[pid_killer].uid;
1890009     pid_t pgrp = proc_table[pid_killer].pgrp;
1890010     int   p;                    // Index inside the process table.
1890011     //
1890012     if (pid_target < -1)
1890013       {
1890014         errset (ESRCH);
1890015         return (-1);
1890016       }
1890017     else if (pid_target == -1)
1890018       {
1890019         if (sig == 0)
1890020           {
1890021             return (0);
1890022           }
1890023         if (euid == 0)
1890024           {
1890025             //
1890026             // Because 'pid_target' is qual to '-1' and the effective
1890027             // user identity is '0', then, all processes,
1890028             // except the kernel and init, will receive the signal.
1890029             //
1890030             // The following scan starts from 2, to preserve the
1890031             // kernel and init processes.
1890032             //
1890033             for (p = 2; p < PROCESS_MAX; p++)
1890034               {
1890035                 if (   proc_table[p].status != PROC_EMPTY
1890036                     && proc_table[p].status != PROC_ZOMBIE)
1890037                   {
1890038                     proc_sig_on (p, sig);
1890039                   }
1890040               }
1890041           }
1890042         else
1890043           {
1890044             //
1890045             // Because 'pid_target' is qual to '-1', but the effective
1890046             // user identity is not '0', then, all processes owned
1890047             // by the same effective user identity, will receive the
1890048             // signal.
1890049             //
1890050             // The following scan starts from 1, to preserve the
1890051             // kernel process.
1890052             //
1890053             for (p = 1; p < PROCESS_MAX; p++)
1890054               {
1890055                 if (   proc_table[p].status != PROC_EMPTY
1890056                     && proc_table[p].status != PROC_ZOMBIE
1890057                     && proc_table[p].uid == euid)
1890058                   {
1890059                     proc_sig_on (p, sig);
1890060                   }
1890061               }
1890062           }
1890063         return (0);
1890064       }
1890065     else if (pid_target == 0)
1890066       {
1890067         if (sig == 0)
1890068           {
1890069             return (0);
1890070           }
```

```
1890071         //
1890072         // The following scan starts from 1, to preserve the
1890073         // kernel process.
1890074         //
1890075         for (p = 1; p < PROCESS_MAX; p++)
1890076           {
1890077             if (   proc_table[p].status != PROC_EMPTY
1890078                 && proc_table[p].status != PROC_ZOMBIE
1890079                 && proc_table[p].pgrp == pgrp)
1890080               {
1890081                 proc_sig_on (p, sig);
1890082               }
1890083           }
1890084         return (0);
1890085       }
1890086     else if (pid_target >= PROCESS_MAX)
1890087       {
1890088         errset (ESRCH);
1890089         return (-1);
1890090       }
1890091     else // (pid_target > 0)
1890092       {
1890093         if (proc_table[pid_target].status == PROC_EMPTY  ||
1890094             proc_table[pid_target].status == PROC_ZOMBIE)
1890095           {
1890096             errset (ESRCH);
1890097             return (-1);
1890098           }
1890099         else if (uid  == proc_table[pid_target].uid   ||
1890100                  uid  == proc_table[pid_target].suid  ||
1890101                  euid == proc_table[pid_target].uid   ||
1890102                  euid == proc_table[pid_target].suid  ||
1890103                  euid == 0)
1890104           {
1890105             if (sig == 0)
1890106               {
1890107                 return (0);
1890108               }
1890109             else
1890110               {
1890111                 proc_sig_on (pid_target, sig);
1890112                 return (0);
1890113               }
1890114           }
1890115         else
1890116           {
1890117             errset (EPERM);
1890118             return (-1);
1890119           }
1890120       }
1890121 }
```

kernel/proc/proc_sys_seteuid.c

«

Si veda la sezione i159.8.24.

```
1900001 #include <kernel/proc.h>
1900002 #include <errno.h>
1900003 //-----------------------------------------------------------------------
1900004 int
1900005 proc_sys_seteuid (pid_t pid, uid_t euid)
1900006 {
1900007   if (proc_table[pid].euid == 0)
1900008     {
1900009       proc_table[pid].euid = euid;
1900010       return (0);
1900011     }
1900012   else if (euid == proc_table[pid].euid)
1900013     {
1900014       return (0);
1900015     }
1900016   else if (euid == proc_table[pid].uid || euid == proc_table[pid].suid)
1900017     {
1900018       proc_table[pid].euid = euid;
1900019       return (0);
1900020     }
1900021   else
1900022     {
1900023       errset (EPERM);
1900024       return (-1);
1900025     }
1900026 }
```

kernel/proc/proc_sys_setuid.c

«

Si veda la sezione i159.8.25.

```
1910001 #include <kernel/proc.h>
1910002 #include <errno.h>
1910003 //-----------------------------------------------------------------------
1910004 int
1910005 proc_sys_setuid (pid_t pid, uid_t uid)
1910006 {
1910007     if (proc_table[pid].euid == 0)
1910008       {
1910009         proc_table[pid].uid  = uid;
1910010         proc_table[pid].euid = uid;
1910011         proc_table[pid].suid = uid;
1910012         return (0);
1910013       }
```



```
1910014     else if (uid == proc_table[pid].euid)
1910015       {
1910016         return (0);
1910017       }
1910018     else if (uid == proc_table[pid].uid || uid == proc_table[pid].suid)
1910019       {
1910020         proc_table[pid].euid = uid;
1910021         return (0);
1910022       }
1910023     else
1910024       {
1910025         errset (EPERM);
1910026         return (-1);
1910027       }
1910028 }
```

kernel/proc/proc_sys_signal.c

«

Si veda la sezione i159.8.26.

```
1920001 #include <kernel/proc.h>
1920002 #include <errno.h>
1920003 //-----------------------------------------------------------------------
1920004 sighandler_t
1920005 proc_sys_signal (pid_t pid, int sig, sighandler_t handler)
1920006 {
1920007     unsigned long int flag = 1L << (sig - 1);
1920008     sighandler_t      previous;
1920009
1920010     if (sig <= 0)
1920011       {
1920012         errset (EINVAL);
1920013         return (SIG_ERR);
1920014       }
1920015
1920016     if (proc_table[pid].sig_ignore & flag)
1920017       {
1920018         previous = SIG_IGN;
1920019       }
1920020     else
1920021       {
1920022         previous = SIG_DFL;
1920023       }
1920024
1920025     if (handler == SIG_DFL)
1920026       {
1920027         proc_table[pid].sig_ignore ^= flag;
1920028         return (previous);
1920029       }
1920030     else if (handler == SIG_IGN)
1920031       {
1920032         proc_table[pid].sig_ignore |= flag;
1920033         return (previous);
1920034       }
1920035     else
1920036       {
1920037         errset (EINVAL);
1920038         return (SIG_ERR);
1920039       }
1920040 }
```

kernel/proc/proc_sys_wait.c

«

Si veda la sezione i159.8.27.

```
1930001 #include <kernel/proc.h>
1930002 #include <errno.h>
1930003 //-----------------------------------------------------------------------
1930004 pid_t
1930005 proc_sys_wait (pid_t pid, int *status)
1930006 {
1930007     pid_t  parent = pid;
1930008     pid_t  child;
1930009     int    child_available = 0;
1930010     //
1930011     // Find a dead child process.
1930012     //
1930013     for (child = 1; child < PROCESS_MAX; child++)
1930014       {
1930015         if (proc_table[child].ppid == parent)
1930016           {
1930017             child_available = 1;        // Child found!
1930018             if (proc_table[child].status == PROC_ZOMBIE)
1930019               {
1930020                 break;                  // It is dead!
1930021               }
1930022           }
1930023       }
1930024     //
1930025     // If the index 'child' is a valid process number,
1930026     // a dead child was found.
1930027     //
1930028     if (child < PROCESS_MAX)
1930029       {
1930030         *status = proc_table[child].ret;
1930031         proc_available (child);
1930032         return (child);
1930033       }
1930034     else
1930035       {
```

```
1930036         if (child_available)
1930037           {
1930038             //
1930039             // There are child, but all alive.
1930040             //
1930041             // Go to sleep.
1930042             //
1930043             proc_table[parent].status         = PROC_SLEEPING;
1930044             proc_table[parent].wakeup_events |= WAKEUP_EVENT_SIGNAL;
1930045             proc_table[parent].wakeup_signal  = SIGCHLD;
1930046             return ((pid_t) 0);
1930047           }
1930048         else
1930049           {
1930050             //
1930051             // There are no child at all.
1930052             //
1930053             errset (ECHILD);
1930054             return ((pid_t) -1);
1930055           }
1930056       }
1930057 }
```

kernel/proc/proc_table.c

«

Si veda la sezione i159.8.7.

```
1940001 #include <kernel/proc.h>
1940002 //---------------------------------------------------------------------------
1940003 proc_t   proc_table[PROCESS_MAX];
```

kernel/proc/sysroutine.c

«

Si veda la sezione i159.8.28.

```
1950001 #include <kernel/proc.h>
1950002 #include <errno.h>
1950003 #include <kernel/k_libc.h>
1950004 //---------------------------------------------------------------------------
1950005 static void sysroutine_error_back (int *number, int *line,
1950006                                    char *file_name);
1950007 //---------------------------------------------------------------------------
1950008 void
1950009 sysroutine (uint16_t *sp, segment_t *segment_d, uint16_t syscallnr,
1950010             uint16_t msg_off, uint16_t msg_size)
1950011 {
1950012   pid_t  pid      = proc_find (*segment_d);
1950013   addr_t msg_addr = address (*segment_d, msg_off);
1950014   //
1950015   // Inbox.
1950016   //
1950017   union {
1950018       sysmsg_chdir_t     chdir;
1950019       sysmsg_chmod_t     chmod;
1950020       sysmsg_chown_t     chown;
1950021       sysmsg_clock_t     clock;
1950022       sysmsg_close_t     close;
1950023       sysmsg_dup_t       dup;
1950024       sysmsg_dup2_t      dup2;
1950025       sysmsg_exec_t      exec;
1950026       sysmsg_exit_t      exit;
1950027       sysmsg_fchmod_t    fchmod;
1950028       sysmsg_fchown_t    fchown;
1950029       sysmsg_fcntl_t     fcntl;
1950030       sysmsg_fork_t      fork;
1950031       sysmsg_fstat_t     fstat;
1950032       sysmsg_kill_t      kill;
1950033       sysmsg_link_t      link;
1950034       sysmsg_lseek_t     lseek;
1950035       sysmsg_mkdir_t     mkdir;
1950036       sysmsg_mknod_t     mknod;
1950037       sysmsg_mount_t     mount;
1950038       sysmsg_open_t      open;
1950039       sysmsg_read_t      read;
1950040       sysmsg_seteuid_t   seteuid;
1950041       sysmsg_setuid_t    setuid;
1950042       sysmsg_signal_t    signal;
1950043       sysmsg_sleep_t     sleep;
1950044       sysmsg_stat_t      stat;
1950045       sysmsg_stime_t     stime;
1950046       sysmsg_time_t      time;
1950047       sysmsg_uarea_t     uarea;
1950048       sysmsg_umask_t     umask;
1950049       sysmsg_umount_t    umount;
1950050       sysmsg_unlink_t    unlink;
1950051       sysmsg_wait_t      wait;
1950052       sysmsg_write_t     write;
1950053       sysmsg_zpchar_t    zpchar;
1950054       sysmsg_zpstring_t  zpstring;
1950055   } msg;
1950056   //
1950057   // Verify if the system call was emitted from kernel code.
1950058   // The kernel can emit only some particular system call:
1950059   //   SYS_NULL, to let other processes run;
1950060   //   SYS_FORK, to let fork itself;
1950061   //   SYS_EXEC, to replace a forked copy of itself.
1950062   //
1950063   if (pid == 0)
1950064     {
1950065       //
```



```
1950066       // This is the kernel code!
1950067       //
1950068       if (   syscallnr != SYS_0
1950069           && syscallnr != SYS_FORK
1950070           && syscallnr != SYS_EXEC)
1950071         {
1950072           k_printf ("kernel panic: the system call %i ", syscallnr);
1950073           k_printf ("was received while running in kernel space!\n");
1950074         }
1950075     }
1950076   //
1950077   // Entering a system call, the kernel variable 'errno' must be
1950078   // reset, otherwise, a previous kernel code error might be returned
1950079   // to the applications.
1950080   //
1950081   errno    = 0;
1950082   errln    = 0;
1950083   errfn[0] = 0;
1950084   //
1950085   // Get message.
1950086   //
1950087   dev_io (pid, DEV_MEM, DEV_READ, msg_addr, &msg, msg_size, NULL);
1950088   //
1950089   // Do the request from the received system call.
1950090   //
1950091   switch (syscallnr)
1950092     {
1950093       case SYS_0:
1950094           break;
1950095       case SYS_CHDIR:
1950096           msg.chdir.ret         = path_chdir (pid, msg.chdir.path);
1950097           sysroutine_error_back (&msg.chdir.errno, &msg.chdir.errln,
1950098                                  msg.chdir.errfn);
1950099           break;
1950100       case SYS_CHMOD:
1950101           msg.chmod.ret         = path_chmod (pid, msg.chmod.path,
1950102                                               msg.chmod.mode);
1950103           sysroutine_error_back (&msg.chmod.errno, &msg.chmod.errln,
1950104                                  msg.chmod.errfn);
1950105           break;
1950106       case SYS_CHOWN:
1950107           msg.chown.ret         = path_chown (pid, msg.chown.path,
1950108                                               msg.chown.uid,
1950109                                               msg.chown.gid);
1950110           sysroutine_error_back (&msg.chown.errno, &msg.chown.errln,
1950111                                  msg.chown.errfn);
1950112           break;
1950113       case SYS_CLOCK:
1950114           msg.clock.ret         = k_clock ();
1950115           break;
1950116       case SYS_CLOSE:
1950117           msg.close.ret         = fd_close (pid, msg.close.fdn);
1950118           sysroutine_error_back (&msg.close.errno, &msg.close.errln,
1950119                                  msg.close.errfn);
1950120           break;
1950121       case SYS_DUP:
1950122           msg.dup.ret           = fd_dup (pid, msg.dup.fdn_old, 0);
1950123           sysroutine_error_back (&msg.dup.errno, &msg.dup.errln,
1950124                                  msg.dup.errfn);
1950125           break;
1950126       case SYS_DUP2:
1950127           msg.dup2.ret          = fd_dup2 (pid, msg.dup2.fdn_old,
1950128                                            msg.dup2.fdn_new);
1950129           sysroutine_error_back (&msg.dup2.errno, &msg.dup2.errln,
1950130                                  msg.dup2.errfn);
1950131           break;
1950132       case SYS_EXEC:
1950133           msg.exec.ret          = proc_sys_exec (sp, segment_d, pid,
1950134                                                  msg.exec.path,
1950135                                                  msg.exec.argc,
1950136                                                  msg.exec.arg_data,
1950137                                                  msg.exec.envc,
1950138                                                  msg.exec.env_data);
1950139           msg.exec.uid          = proc_table[pid].uid;
1950140           msg.exec.euid         = proc_table[pid].euid;
1950141           sysroutine_error_back (&msg.exec.errno, &msg.exec.errln,
1950142                                  msg.exec.errfn);
1950143           break;
1950144       case SYS_EXIT:
1950145           if (pid == 0)
1950146             {
1950147               k_printf ("kernel alert: "
1950148                         "the kernel cannot exit!\n");
1950149             }
1950150           else
1950151             {
1950152               proc_sys_exit (pid, msg.exit.status);
1950153             }
1950154           break;
1950155       case SYS_FCHMOD:
1950156           msg.fchmod.ret        = fd_chmod (pid, msg.fchmod.fdn,
1950157                                             msg.fchmod.mode);
1950158           sysroutine_error_back (&msg.fchmod.errno, &msg.fchmod.errln,
1950159                                  msg.fchmod.errfn);
1950160           break;
1950161       case SYS_FCHOWN:
1950162           msg.fchown.ret        = fd_chown (pid, msg.fchown.fdn,
1950163                                             msg.fchown.uid,
1950164                                             msg.fchown.gid);
1950165           sysroutine_error_back (&msg.fchown.errno, &msg.fchown.errln,
1950166                                  msg.fchown.errfn);
```

```
1950167             break;
1950168         case SYS_FCNTL:
1950169             msg.fcntl.ret         = fd_fcntl (pid, msg.fcntl.fdn,
1950170                                               msg.fcntl.cmd,
1950171                                               msg.fcntl.arg);
1950172             sysroutine_error_back (&msg.fcntl.errno, &msg.fcntl.errln,
1950173                                    msg.fcntl.errfn);
1950174             break;
1950175         case SYS_FORK:
1950176             msg.fork.ret          = proc_sys_fork (pid, *sp);
1950177             sysroutine_error_back (&msg.fork.errno, &msg.fork.errln,
1950178                                    msg.fork.errfn);
1950179             break;
1950180         case SYS_FSTAT:
1950181             msg.fstat.ret         = fd_stat (pid, msg.fstat.fdn,
1950182                                              &msg.fstat.stat);
1950183             sysroutine_error_back (&msg.fstat.errno, &msg.fstat.errln,
1950184                                    msg.fstat.errfn);
1950185             break;
1950186         case SYS_KILL:
1950187             msg.kill.ret          = proc_sys_kill (pid, msg.kill.pid,
1950188                                                    msg.kill.signal);
1950189             sysroutine_error_back (&msg.kill.errno, &msg.kill.errln,
1950190                                    msg.kill.errfn);
1950191             break;
1950192         case SYS_LINK:
1950193             msg.link.ret          = path_link (pid, msg.link.path_old,
1950194                                                msg.link.path_new);
1950195             sysroutine_error_back (&msg.link.errno, &msg.link.errln,
1950196                                    msg.link.errfn);
1950197             break;
1950198         case SYS_LSEEK:
1950199             msg.lseek.ret         = fd_lseek (pid, msg.lseek.fdn,
1950200                                               msg.lseek.offset,
1950201                                               msg.lseek.whence);
1950202             sysroutine_error_back (&msg.lseek.errno, &msg.lseek.errln,
1950203                                    msg.lseek.errfn);
1950204             break;
1950205         case SYS_MKDIR:
1950206             msg.mkdir.ret         = path_mkdir (pid, msg.mkdir.path,
1950207                                                 msg.mkdir.mode);
1950208             sysroutine_error_back (&msg.mkdir.errno, &msg.mkdir.errln,
1950209                                    msg.mkdir.errfn);
1950210             break;
1950211         case SYS_MKNOD:
1950212             msg.mknod.ret         = path_mknod (pid, msg.mknod.path,
1950213                                                 msg.mknod.mode,
1950214                                                 msg.mknod.device);
1950215             sysroutine_error_back (&msg.mknod.errno, &msg.mknod.errln,
1950216                                    msg.mknod.errfn);
1950217             break;
1950218         case SYS_MOUNT:
1950219             msg.mount.ret         = path_mount (pid, msg.mount.path_dev,
1950220                                                 msg.mount.path_mnt,
1950221                                                 msg.mount.options);
1950222             sysroutine_error_back (&msg.mount.errno, &msg.mount.errln,
1950223                                    msg.mount.errfn);
1950224             break;
1950225         case SYS_OPEN:
1950226             msg.open.ret          = fd_open (pid, msg.open.path,
1950227                                              msg.open.flags,
1950228                                              msg.open.mode);
1950229             sysroutine_error_back (&msg.open.errno, &msg.open.errln,
1950230                                    msg.open.errfn);
1950231             break;
1950232         case SYS_PGRP:
1950233             proc_table[pid].pgrp          = pid;
1950234             break;
1950235         case SYS_READ:
1950236             msg.read.ret          = fd_read (pid, msg.read.fdn,
1950237                                              msg.read.buffer,
1950238                                              msg.read.count,
1950239                                              &msg.read.eof);
1950240             sysroutine_error_back (&msg.read.errno, &msg.read.errln,
1950241                                    msg.read.errfn);
1950242             break;
1950243         case SYS_SETEUID:
1950244             msg.seteuid.ret       = proc_sys_seteuid (pid,
1950245                                                       msg.seteuid.euid);
1950246             msg.seteuid.euid      = proc_table[pid].euid;
1950247             sysroutine_error_back (&msg.seteuid.errno, &msg.seteuid.errln,
1950248                                    msg.seteuid.errfn);
1950249             break;
1950250         case SYS_SETUID:
1950251             msg.setuid.ret        = proc_sys_setuid (pid,
1950252                                                      msg.setuid.euid);
1950253             msg.setuid.uid        = proc_table[pid].uid;
1950254             msg.setuid.euid       = proc_table[pid].euid;
1950255             msg.setuid.suid       = proc_table[pid].suid;
1950256             sysroutine_error_back (&msg.setuid.errno, &msg.setuid.errln,
1950257                                    msg.setuid.errfn);
1950258             break;
1950259         case SYS_SIGNAL:
1950260             msg.signal.ret        = proc_sys_signal (pid,
1950261                                                      msg.signal.signal,
1950262                                                     msg.signal.handler);
1950263             sysroutine_error_back (&msg.signal.errno, &msg.signal.errln,
1950264                                    msg.signal.errfn);
1950265             break;
1950266         case SYS_SLEEP:
1950267             proc_table[pid].status        = PROC_SLEEPING;
```



```
1950268             proc_table[pid].ret           = 0;
1950269             proc_table[pid].wakeup_events = msg.sleep.events;
1950270             proc_table[pid].wakeup_signal = msg.sleep.signal;
1950271             proc_table[pid].wakeup_timer  = msg.sleep.seconds;
1950272             break;
1950273         case SYS_STAT:
1950274             msg.stat.ret          = path_stat (pid, msg.stat.path,
1950275                                                &msg.stat.stat);
1950276             sysroutine_error_back (&msg.stat.errno, &msg.stat.errln,
1950277                                    msg.stat.errfn);
1950278             break;
1950279         case SYS_STIME:
1950280             msg.stime.ret         = k_stime (&msg.stime.timer);
1950281             break;
1950282         case SYS_TIME:
1950283             msg.time.ret          = k_time (NULL);
1950284             break;
1950285         case SYS_UAREA:
1950286             msg.uarea.uid         = proc_table[pid].uid;
1950287             msg.uarea.euid        = proc_table[pid].euid;
1950288             msg.uarea.pid         = pid;
1950289             msg.uarea.ppid        = proc_table[pid].ppid;
1950290             msg.uarea.pgrp        = proc_table[pid].pgrp;
1950291             msg.uarea.umask       = proc_table[pid].umask;
1950292             strncpy (msg.uarea.path_cwd,
1950293                      proc_table[pid].path_cwd, PATH_MAX);
1950294             break;
1950295         case SYS_UMASK:
1950296             msg.umask.ret         = proc_table[pid].umask;
1950297             proc_table[pid].umask = (msg.umask.umask & 00777);
1950298             break;
1950299         case SYS_UMOUNT:
1950300             msg.umount.ret        = path_umount (pid,
1950301                                                  msg.umount.path_mnt);
1950302             sysroutine_error_back (&msg.umount.errno, &msg.umount.errln,
1950303                                    msg.umount.errfn);
1950304             break;
1950305         case SYS_UNLINK:
1950306             msg.unlink.ret        = path_unlink (pid, msg.unlink.path);
1950307             sysroutine_error_back (&msg.unlink.errno, &msg.unlink.errln,
1950308                                    msg.unlink.errfn);
1950309             break;
1950310         case SYS_WAIT:
1950311             msg.wait.ret          = proc_sys_wait (pid,
1950312                                                    &msg.wait.status);
1950313             sysroutine_error_back (&msg.wait.errno, &msg.wait.errln,
1950314                                    msg.wait.errfn);
1950315             break;
1950316         case SYS_WRITE:
1950317             msg.write.ret         = fd_write (pid, msg.write.fdn,
1950318                                               msg.write.buffer,
1950319                                               msg.write.count);
1950320             sysroutine_error_back (&msg.write.errno, &msg.write.errln,
1950321                                    msg.write.errfn);
1950322             break;
1950323         case SYS_ZPCHAR:
1950324             dev_io (pid, DEV_TTY, DEV_WRITE, 0L, &msg.zpchar.c,
1950325                     1, NULL);
1950326             break;
1950327         case SYS_ZPSTRING:
1950328             dev_io (pid, DEV_TTY, DEV_WRITE, 0L,
1950329                     msg.zpstring.string,
1950330                     strlen (msg.zpstring.string), NULL);
1950331             break;
1950332         default:
1950333             k_printf ("kernel alert: unknown system call %i!\n",
1950334                       syscallnr);
1950335             break;
1950336       }
1950337     //
1950338     // Return value with a message back.
1950339     //
1950340     dev_io (pid, DEV_MEM, DEV_WRITE, msg_addr, &msg, msg_size, NULL);
1950341     //
1950342     // Continue with the scheduler.
1950343     //
1950344     proc_scheduler (sp, segment_d);
1950345   }
1950346   //---------------------------------------------------------------------
1950347   static void
1950348   sysroutine_error_back (int *number, int *line, char *file_name)
1950349   {
1950350     *number = errno;
1950351     *line   = errln;
1950352     strncpy (file_name, errfn, PATH_MAX);
1950353     file_name[PATH_MAX-1] = 0;
1950354   }
```

os16: «kernel/tty.h»

Si veda la sezione u0.9.

```
1960001   #ifndef _KERNEL_TTY_H
1960002   #define _KERNEL_TTY_H  1
1960003
1960004   #include <stddef.h>
1960005   #include <stdint.h>
1960006   #include <sys/types.h>
1960007   #include <kernel/ibm_i86.h>
1960008
1960009   //---------------------------------------------------------------------
```

```
1960010 #define         TTYS_CONSOLE    IBM_I86_VIDEO_PAGES
1960011 #define         TTYS_SERIAL     0
1960012 #define         TTYS_TOTAL      (TTYS_CONSOLE + TTYS_SERIAL)
1960013 //---------------------------------------------------------------------
1960014 #define         TTY_OK          0
1960015 #define         TTY_LOST_KEY    1
1960016 //---------------------------------------------------------------------
1960017 typedef struct {
1960018     dev_t       device;
1960019     pid_t       pgrp;            // Process group.
1960020     int         key;             // Last pressed key.
1960021     int         status;          // 0 = ok, 1 = lost typed character.
1960022 } tty_t;
1960023 //---------------------------------------------------------------------
1960024 extern tty_t tty_table[TTYS_TOTAL];
1960025 //---------------------------------------------------------------------
1960026 tty_t *tty_reference (dev_t device);
1960027 dev_t  tty_console   (dev_t device);
1960028 int    tty_read      (dev_t device);
1960029 void   tty_write     (dev_t device, int c);
1960030 void   tty_init      (void);
1960031
1960032 #endif
```

kernel/tty/tty_console.c

«

Si veda la sezione u0.9.

```
1970001 #include <sys/os16.h>
1970002 #include <kernel/tty.h>
1970003 //---------------------------------------------------------------------
1970004 dev_t
1970005 tty_console (dev_t device)
1970006 {
1970007     static dev_t device_active = DEV_CONSOLE0;  // First time.
1970008            dev_t device_previous;
1970009     //
1970010     // Check if it required only the current device.
1970011     //
1970012     if (device == 0)
1970013       {
1970014         return (device_active);
1970015       }
1970016     //
1970017     // Fix if the device is not valid.
1970018     //
1970019     if (device > DEV_CONSOLE3 || device < DEV_CONSOLE0)
1970020       {
1970021         device = DEV_CONSOLE0;
1970022       }
1970023     //
1970024     // Update.
1970025     //
1970026     device_previous = device_active;
1970027     device_active   = device;
1970028     //
1970029     // Switch.
1970030     //
1970031     con_select (device_active & 0x00FF);
1970032     //
1970033     // Return previous device value.
1970034     //
1970035     return (device_previous);
1970036 }
```

kernel/tty/tty_init.c

«

Si veda la sezione u0.9.

```
1980001 #include <sys/os16.h>
1980002 #include <kernel/tty.h>
1980003 //---------------------------------------------------------------------
1980004 void
1980005 tty_init (void)
1980006 {
1980007     int page;   // console page.
1980008     //
1980009     // Console initialization: console pages correspond to the first
1980010     // terminal items.
1980011     //
1980012     for (page = 0; page < TTYS_CONSOLE; page++)
1980013       {
1980014         tty_table[page].device = DEV_CONSOLE0 + page;
1980015         tty_table[page].pgrp   = 0;
1980016         tty_table[page].key    = 0;
1980017         tty_table[page].status = TTY_OK;
1980018       }
1980019     //
1980020     // Set video mode.
1980021     //
1980022     con_init ();
1980023     //
1980024     // Select the first console.
1980025     //
1980026     tty_console (DEV_CONSOLE0);
1980027     //
1980028     // Nothing else to configure (only consoles are available).
1980029     //
1980030     return;
1980031 }
```


kernel/tty/tty_read.c

«

Si veda la sezione u0.9.

```
1990001 #include <sys/os16.h>
1990002 #include <kernel/tty.h>
1990003 #include <kernel/k_libc.h>
1990004 //---------------------------------------------------------------------
1990005 int
1990006 tty_read (dev_t device)
1990007 {
1990008     tty_t   *tty;
1990009     int      key;
1990010     //
1990011     tty = tty_reference (device);
1990012     if (tty == NULL)
1990013       {
1990014         k_printf ("kernel alert: cannot find terminal device "
1990015                   "0x%04x!\n", (int) device);
1990016         //
1990017         return (0);
1990018       }
1990019     //
1990020     // Read key and remove from the source.
1990021     //
1990022     key      = tty->key;
1990023     tty->key = 0;
1990024     //
1990025     // Return the key.
1990026     //
1990027     return (key);
1990028
1990029 }
```

kernel/tty/tty_reference.c

«

Si veda la sezione u0.9.

```
2000001 #include <kernel/tty.h>
2000002 //---------------------------------------------------------------------
2000003 tty_t *
2000004 tty_reference (dev_t device)
2000005 {
2000006     int t;              // Terminal index.
2000007     //
2000008     // If device is zero, a reference to the whole table is returned.
2000009     //
2000010     if (device == 0)
2000011       {
2000012         return (tty_table);
2000013       }
2000014     //
2000015     // Otherwise, a scan is made to find the selected device.
2000016     //
2000017     for (t = 0; t < TTYS_TOTAL; t++)
2000018       {
2000019         if (tty_table[t].device == device)
2000020           {
2000021             //
2000022             // Device found. Return the pointer.
2000023             //
2000024             return (&tty_table[t]);
2000025           }
2000026       }
2000027     //
2000028     // No device found!
2000029     //
2000030     return (NULL);
2000031 }
```

kernel/tty/tty_table.c

«

Si veda la sezione u0.9.

```
2010001 #include <kernel/tty.h>
2010002 //---------------------------------------------------------------------
2010003 tty_t tty_table[TTYS_TOTAL];
```

kernel/tty/tty_write.c

«

Si veda la sezione u0.9.

```
2020001 #include <sys/os16.h>
2020002 #include <kernel/tty.h>
2020003 //---------------------------------------------------------------------
2020004 void
2020005 tty_write (dev_t device, int c)
2020006 {
2020007     int   console;
2020008     //
2020009     if ((device & 0xFF00) == (DEV_CONSOLE_MAJOR << 8))
2020010       {
2020011         console = (device & 0x00FF);
2020012         con_putc (console, c);
2020013       }
2020014 }
```

os16: file isolati della directory «lib/» 1735
lib/NULL.h 1735
lib/SEEK.h 1735
lib/_Bool.h 1735
lib/clock_t.h 1736
lib/const.h 1736
lib/ctype.h 1736
lib/inttypes.h 1736
lib/limits.h 1738
lib/ptrdiff_t.h 1738
lib/restrict.h 1738
lib/size_t.h 1739
lib/stdarg.h 1739
lib/stdbool.h 1739
lib/stddef.h 1739
lib/stdint.h 1739
lib/time_t.h 1740
lib/wchar_t.h 1740
os16: «lib/dirent.h» 1741
lib/dirent/DIR.c 1741
lib/dirent/closedir.c 1741
lib/dirent/opendir.c 1742
lib/dirent/readdir.c 1743
lib/dirent/rewinddir.c 1744
os16: «lib/errno.h» 1744
lib/errno/errno.c 1746
os16: «lib/fcntl.h» 1747
lib/fcntl/creat.c 1748
lib/fcntl/fcntl.c 1748
lib/fcntl/open.c 1748
os16: «lib/grp.h» 1749
lib/grp/getgrgid.c 1749
lib/grp/getgrnam.c 1749
os16: «lib/libgen.h» 1749
lib/libgen/basename.c 1749
lib/libgen/dirname.c 1750
os16: «lib/pwd.h» 1751
lib/pwd/pwent.c 1751
os16: «lib/signal.h» 1752
lib/signal/kill.c 1753
lib/signal/signal.c 1753
os16: «lib/stdio.h» 1753
lib/stdio/FILE.c 1754
lib/stdio/clearerr.c 1755
lib/stdio/fclose.c 1755
lib/stdio/feof.c 1755
lib/stdio/ferror.c 1755
lib/stdio/fflush.c 1755
lib/stdio/fgetc.c 1755
lib/stdio/fgetpos.c 1756
lib/stdio/fgets.c 1756
lib/stdio/fileno.c 1756
lib/stdio/fopen.c 1757

lib/stdio/fprintf.c 1757
lib/stdio/fputc.c 1758
lib/stdio/fputs.c 1758
lib/stdio/fread.c 1758
lib/stdio/freopen.c 1758
lib/stdio/fscanf.c 1759
lib/stdio/fseek.c 1759
lib/stdio/fseeko.c 1759
lib/stdio/fsetpos.c 1760
lib/stdio/ftell.c 1760
lib/stdio/ftello.c 1760
lib/stdio/fwrite.c 1760
lib/stdio/getchar.c 1760
lib/stdio/gets.c 1761
lib/stdio/perror.c 1761
lib/stdio/printf.c 1762
lib/stdio/puts.c 1762
lib/stdio/rewind.c 1762
lib/stdio/scanf.c 1762
lib/stdio/setbuf.c 1762
lib/stdio/setvbuf.c 1762
lib/stdio/snprintf.c 1763
lib/stdio/sprintf.c 1763
lib/stdio/sscanf.c 1763
lib/stdio/vfprintf.c 1763
lib/stdio/vfscanf.c 1764
lib/stdio/vfsscanf.c 1764
lib/stdio/vprintf.c 1777
lib/stdio/vscanf.c 1778
lib/stdio/vsnprintf.c 1778
lib/stdio/vsprintf.c 1788
lib/stdio/vsscanf.c 1788
os16: «lib/stdlib.h» 1788
lib/stdlib/_Exit.c 1789
lib/stdlib/abort.c 1789
lib/stdlib/abs.c 1789
lib/stdlib/alloc.c 1790
lib/stdlib/atexit.c 1792
lib/stdlib/atoi.c 1792
lib/stdlib/atol.c 1793
lib/stdlib/div.c 1793
lib/stdlib/environment.c 1793
lib/stdlib/exit.c 1794
lib/stdlib/getenv.c 1794
lib/stdlib/labs.c 1795
lib/stdlib/ldiv.c 1795
lib/stdlib/putenv.c 1795
lib/stdlib/qsort.c 1796
lib/stdlib/rand.c 1798
lib/stdlib/setenv.c 1798
lib/stdlib/strtol.c 1799
lib/stdlib/strtoul.c 1801
lib/stdlib/unsetenv.c 1801
os16: «lib/string.h» 1802
lib/string/memccpy.c 1803
lib/string/memchr.c 1803
lib/string/memcmp.c 1803

lib/string/memcpy.c 1803
lib/string/memmove.c 1803
lib/string/memset.c 1804
lib/string/strcat.c 1804
lib/string/strchr.c 1804
lib/string/strcmp.c 1805
lib/string/strcoll.c 1805
lib/string/strcpy.c 1805
lib/string/strcspn.c 1805
lib/string/strdup.c 1805
lib/string/strerror.c 1806
lib/string/strlen.c 1807
lib/string/strncat.c 1807
lib/string/strncmp.c 1807
lib/string/strncpy.c 1808
lib/string/strpbrk.c 1808
lib/string/strrchr.c 1808
lib/string/strspn.c 1808
lib/string/strstr.c 1809
lib/string/strtok.c 1809
lib/string/strxfrm.c 1810
os16: «lib/sys/os16.h» 1811
lib/sys/os16/_bp.s 1815
lib/sys/os16/_cs.s 1816
lib/sys/os16/_ds.s 1816
lib/sys/os16/_es.s 1816
lib/sys/os16/_seg_d.s 1816
lib/sys/os16/_seg_i.s 1816
lib/sys/os16/_sp.s 1816
lib/sys/os16/_ss.s 1817
lib/sys/os16/heap_clear.c 1817
lib/sys/os16/heap_min.c 1817
lib/sys/os16/input_line.c 1817
lib/sys/os16/mount.c 1818
lib/sys/os16/namep.c 1818
lib/sys/os16/process_info.c 1820
lib/sys/os16/sys.s 1820
lib/sys/os16/umount.c 1820
lib/sys/os16/z_perror.c 1820
lib/sys/os16/z_printf.c 1821
lib/sys/os16/z_putchar.c 1821
lib/sys/os16/z_puts.c 1821
lib/sys/os16/z_vprintf.c 1821
os16: «lib/sys/stat.h» 1821
lib/sys/stat/chmod.c 1823
lib/sys/stat/fchmod.c 1823
lib/sys/stat/fstat.c 1823
lib/sys/stat/mkdir.c 1824
lib/sys/stat/mknod.c 1824
lib/sys/stat/stat.c 1824
lib/sys/stat/umask.c 1825
os16: «lib/sys/types.h» 1825
lib/sys/types/major.c 1825
lib/sys/types/makedev.c 1825
lib/sys/types/minor.c 1826
os16: «lib/sys/wait.h» 1826
lib/sys/wait/wait.c 1826

os16: «lib/time.h» 1826
lib/time/asctime.c 1827
lib/time/clock.c 1828
lib/time/gmtime.c 1828
lib/time/mktime.c 1830
lib/time/stime.c 1831
lib/time/time.c 1831
os16: «lib/unistd.h» 1831
lib/unistd/_exit.c 1832
lib/unistd/access.c 1832
lib/unistd/chdir.c 1833
lib/unistd/chown.c 1833
lib/unistd/close.c 1834
lib/unistd/dup.c 1834
lib/unistd/dup2.c 1834
lib/unistd/environ.c 1834
lib/unistd/execl.c 1834
lib/unistd/execle.c 1835
lib/unistd/execlp.c 1835
lib/unistd/execv.c 1836
lib/unistd/execve.c 1836
lib/unistd/execvp.c 1837
lib/unistd/fchdir.c 1837
lib/unistd/fchown.c 1837
lib/unistd/fork.c 1837
lib/unistd/getcwd.c 1838
lib/unistd/geteuid.c 1838
lib/unistd/getopt.c 1839
lib/unistd/getpgrp.c 1841
lib/unistd/getpid.c 1841
lib/unistd/getppid.c 1841
lib/unistd/getuid.c 1842
lib/unistd/isatty.c 1842
lib/unistd/link.c 1842
lib/unistd/lseek.c 1842
lib/unistd/read.c 1843
lib/unistd/rmdir.c 1843
lib/unistd/seteuid.c 1844
lib/unistd/setpgrp.c 1844
lib/unistd/setuid.c 1844
lib/unistd/sleep.c 1845
lib/unistd/ttyname.c 1845
lib/unistd/unlink.c 1846
lib/unistd/write.c 1846
os16: «lib/utime.h» 1847
lib/utime/utime.c 1847

abort.c 1789 abs.c 1789 access.c 1832 alloc.c 1790 asctime.c 1827 atexit.c 1792 atoi.c 1792 atol.c 1793 basename.c 1749 chdir.c 1833 chmod.c 1823 chown.c 1833 clearerr.c 1755 clock.c 1828 clock_t.h 1736 close.c 1834 closedir.c 1741 const.h 1736 creat.c 1748 ctype.h 1736 DIR.c 1741 dirent.h 1741 dirname.c 1750 div.c 1793 dup.c 1834 dup2.c 1834 environ.c 1834 environment.c 1793 errno.c 1746 errno.h 1744 execl.c 1834 execle.c 1835 execlp.c 1835 execv.c 1836 execve.c 1836 execvp.c 1837 exit.c 1794 fchdir.c 1837 fchmod.c 1823 fchown.c 1837 fclose.c 1755 fcntl.c 1748 fcntl.h 1747 feof.c 1755 ferror.c 1755 fflush.c 1755 fgetc.c 1755 fgetpos.c 1756 fgets.c 1756 FILE.c 1754 fileno.c 1756 fopen.c 1757 fork.c 1837 fprintf.c 1757 fputc.c 1758 fputs.c 1758 fread.c 1758 freopen.c 1758 fscanf.c 1759 fseek.c 1759 fseeko.c 1759 fsetpos.c 1760 fstat.c 1823 ftell.c 1760 ftello.c 1760 fwrite.c 1760 getchar.c 1760 getcwd.c 1838 getenv.c 1794 geteuid.c 1838 getgrgid.c 1749 getgrnam.c 1749 getopt.c 1839 getpgrp.c 1841 getpid.c 1841 getppid.c 1841 gets.c 1761 getuid.c 1842 gmtime.c 1828 grp.h 1749 heap_clear.c 1817 heap_min.c 1817 input_line.c 1817 inttypes.h 1736 isatty.c 1842 kill.c 1753 labs.c 1795 ldiv.c 1795 libgen.h 1749 limits.h 1738 link.c 1842 lseek.c 1842 major.c 1825 makedev.c 1825 memccpy.c 1803 memchr.c 1803 memcmp.c 1803 memcpy.c 1803 memmove.c 1803 memset.c 1804 minor.c 1826 mkdir.c 1824 mknod.c 1824 mktime.c 1830 mount.c 1818 namep.c 1818 NULL.h 1735 open.c 1748 opendir.c 1742 os16.h 1811 perror.c 1761 printf.c 1762 process_info.c 1820 ptrdiff_t.h 1738 putenv.c 1795 puts.c 1762 pwd.h 1751 pwent.c 1751 qsort.c 1796 rand.c 1798 read.c 1843 readdir.c 1743 restrict.h 1738 rewind.c 1762 rewinddir.c 1744 rmdir.c 1843 scanf.c 1762 SEEK.h 1735 setbuf.c 1762 setenv.c 1798 seteuid.c 1844 setpgrp.c 1844 setuid.c 1844 setvbuf.c 1762 signal.c 1753 signal.h 1752 size_t.h 1739 sleep.c 1845 snprintf.c 1763 sprintf.c 1763 sscanf.c 1763 stat.c 1824 stat.h 1821 stdarg.h 1739 stdbool.h 1739 stddef.h 1739 stdint.h 1739 stdio.h 1753 stdlib.h 1788 stime.c 1831 strcat.c 1804 strchr.c 1804 strcmp.c 1805 strcoll.c 1805 strcpy.c 1805 strcspn.c 1805 strdup.c 1805 strerror.c 1806 string.h 1802 strlen.c 1807 strncat.c 1807 strncmp.c 1807 strncpy.c 1808 strpbrk.c 1808 strrchr.c 1808 strspn.c 1808 strstr.c 1809 strtok.c 1809 strtol.c 1799 strtoul.c 1801 strxfrm.c 1810 sys.s 1820 time.c 1831 time.h 1826 time_t.h 1740 ttyname.c 1845 types.h 1825 umask.c 1825 umount.c 1820 unistd.h 1831 unlink.c 1846 unsetenv.c 1801 utime.c 1847 utime.h 1847 vfprintf.c 1763 vfscanf.c 1764 vfsscanf.c 1764 vprintf.c 1777 vscanf.c 1778 vsnprintf.c 1778 vsprintf.c 1788 vsscanf.c 1788 wait.c 1826 wait.h 1826 wchar_t.h 1740 write.c 1846 z_perror.c 1820 z_printf.c 1821 z_putchar.c 1821 z_puts.c 1821 z_vprintf.c 1821 _Bool.h 1735 _bp.s 1815 _cs.s 1816 _ds.s 1816 _es.s 1816 _exit.c 1832 _Exit.c 1789 _seg_d.s 1816 _seg_i.s 1816 _sp.s 1816 _ss.s 1817

os16: file isolati della directory «lib/» 1735
lib/NULL.h 1735
lib/SEEK.h 1735
lib/_Bool.h 1735
lib/clock_t.h 1736
lib/const.h 1736
lib/ctype.h 1736
lib/inttypes.h 1736
lib/limits.h 1738
lib/ptrdiff_t.h 1738
lib/restrict.h 1738
lib/size_t.h 1739
lib/stdarg.h 1739
lib/stdbool.h 1739
lib/stddef.h 1739

lib/stdint.h 1739
lib/time_t.h 1740
lib/wchar_t.h 1740
os16: «lib/dirent.h» 1741
lib/dirent/DIR.c 1741
lib/dirent/closedir.c 1741
lib/dirent/opendir.c 1742
lib/dirent/readdir.c 1743
lib/dirent/rewinddir.c 1744
os16: «lib/errno.h» 1744
lib/errno/errno.c 1746
os16: «lib/fcntl.h» 1747
lib/fcntl/creat.c 1748
lib/fcntl/fcntl.c 1748
lib/fcntl/open.c 1748
os16: «lib/grp.h» 1749
lib/grp/getgrgid.c 1749
lib/grp/getgrnam.c 1749
os16: «lib/libgen.h» 1749
lib/libgen/basename.c 1749
lib/libgen/dirname.c 1750
os16: «lib/pwd.h» 1751
lib/pwd/pwent.c 1751
os16: «lib/signal.h» 1752
lib/signal/kill.c 1753
lib/signal/signal.c 1753
os16: «lib/stdio.h» 1753
lib/stdio/FILE.c 1754
lib/stdio/clearerr.c 1755
lib/stdio/fclose.c 1755
lib/stdio/feof.c 1755
lib/stdio/ferror.c 1755
lib/stdio/fflush.c 1755
lib/stdio/fgetc.c 1755
lib/stdio/fgetpos.c 1756
lib/stdio/fgets.c 1756
lib/stdio/fileno.c 1756
lib/stdio/fopen.c 1757
lib/stdio/fprintf.c 1757
lib/stdio/fputc.c 1758
lib/stdio/fputs.c 1758
lib/stdio/fread.c 1758
lib/stdio/freopen.c 1758
lib/stdio/fscanf.c 1759
lib/stdio/fseek.c 1759
lib/stdio/fseeko.c 1759
lib/stdio/fsetpos.c 1760
lib/stdio/ftell.c 1760
lib/stdio/ftello.c 1760
lib/stdio/fwrite.c 1760
lib/stdio/getchar.c 1760
lib/stdio/gets.c 1761
lib/stdio/perror.c 1761
lib/stdio/printf.c 1762
lib/stdio/puts.c 1762

lib/stdio/rewind.c 1762
lib/stdio/scanf.c 1762
lib/stdio/setbuf.c 1762
lib/stdio/setvbuf.c 1762
lib/stdio/snprintf.c 1763
lib/stdio/sprintf.c 1763
lib/stdio/sscanf.c 1763
lib/stdio/vfprintf.c 1763
lib/stdio/vfscanf.c 1764
lib/stdio/vfsscanf.c 1764
lib/stdio/vprintf.c 1777
lib/stdio/vscanf.c 1778
lib/stdio/vsnprintf.c 1778
lib/stdio/vsprintf.c 1788
lib/stdio/vsscanf.c 1788
os16: «lib/stdlib.h» 1788
lib/stdlib/_Exit.c 1789
lib/stdlib/abort.c 1789
lib/stdlib/abs.c 1789
lib/stdlib/alloc.c 1790
lib/stdlib/atexit.c 1792
lib/stdlib/atoi.c 1792
lib/stdlib/atol.c 1793
lib/stdlib/div.c 1793
lib/stdlib/environment.c 1793
lib/stdlib/exit.c 1794
lib/stdlib/getenv.c 1794
lib/stdlib/labs.c 1795
lib/stdlib/ldiv.c 1795
lib/stdlib/putenv.c 1795
lib/stdlib/qsort.c 1796
lib/stdlib/rand.c 1798
lib/stdlib/setenv.c 1798
lib/stdlib/strtol.c 1799
lib/stdlib/strtoul.c 1801
lib/stdlib/unsetenv.c 1801
os16: «lib/string.h» 1802
lib/string/memccpy.c 1803
lib/string/memchr.c 1803
lib/string/memcmp.c 1803
lib/string/memcpy.c 1803
lib/string/memmove.c 1803
lib/string/memset.c 1804
lib/string/strcat.c 1804
lib/string/strchr.c 1804
lib/string/strcmp.c 1805
lib/string/strcoll.c 1805
lib/string/strcpy.c 1805
lib/string/strcspn.c 1805
lib/string/strdup.c 1805
lib/string/strerror.c 1806
lib/string/strlen.c 1807
lib/string/strncat.c 1807
lib/string/strncmp.c 1807
lib/string/strncpy.c 1808
lib/string/strpbrk.c 1808
lib/string/strrchr.c 1808

lib/string/strspn.c 1808
lib/string/strstr.c 1809
lib/string/strtok.c 1809
lib/string/strxfrm.c 1810
os16: «lib/sys/os16.h» 1811
lib/sys/os16/_bp.s 1815
lib/sys/os16/_cs.s 1816
lib/sys/os16/_ds.s 1816
lib/sys/os16/_es.s 1816
lib/sys/os16/_seg_d.s 1816
lib/sys/os16/_seg_i.s 1816
lib/sys/os16/_sp.s 1816
lib/sys/os16/_ss.s 1817
lib/sys/os16/heap_clear.c 1817
lib/sys/os16/heap_min.c 1817
lib/sys/os16/input_line.c 1817
lib/sys/os16/mount.c 1818
lib/sys/os16/namep.c 1818
lib/sys/os16/process_info.c 1820
lib/sys/os16/sys.s 1820
lib/sys/os16/umount.c 1820
lib/sys/os16/z_perror.c 1820
lib/sys/os16/z_printf.c 1821
lib/sys/os16/z_putchar.c 1821
lib/sys/os16/z_puts.c 1821
lib/sys/os16/z_vprintf.c 1821
os16: «lib/sys/stat.h» 1821
lib/sys/stat/chmod.c 1823
lib/sys/stat/fchmod.c 1823
lib/sys/stat/fstat.c 1823
lib/sys/stat/mkdir.c 1824
lib/sys/stat/mknod.c 1824
lib/sys/stat/stat.c 1824
lib/sys/stat/umask.c 1825
os16: «lib/sys/types.h» 1825
lib/sys/types/major.c 1825
lib/sys/types/makedev.c 1825
lib/sys/types/minor.c 1826
os16: «lib/sys/wait.h» 1826
lib/sys/wait/wait.c 1826
os16: «lib/time.h» 1826
lib/time/asctime.c 1827
lib/time/clock.c 1828
lib/time/gmtime.c 1828
lib/time/mktime.c 1830
lib/time/stime.c 1831
lib/time/time.c 1831
os16: «lib/unistd.h» 1831
lib/unistd/_exit.c 1832
lib/unistd/access.c 1832
lib/unistd/chdir.c 1833
lib/unistd/chown.c 1833
lib/unistd/close.c 1834
lib/unistd/dup.c 1834
lib/unistd/dup2.c 1834
lib/unistd/environ.c 1834

lib/unistd/execl.c 1834
lib/unistd/execle.c 1835
lib/unistd/execlp.c 1835
lib/unistd/execv.c 1836
lib/unistd/execve.c 1836
lib/unistd/execvp.c 1837
lib/unistd/fchdir.c 1837
lib/unistd/fchown.c 1837
lib/unistd/fork.c 1837
lib/unistd/getcwd.c 1838
lib/unistd/geteuid.c 1838
lib/unistd/getopt.c 1839
lib/unistd/getpgrp.c 1841
lib/unistd/getpid.c 1841
lib/unistd/getppid.c 1841
lib/unistd/getuid.c 1842
lib/unistd/isatty.c 1842
lib/unistd/link.c 1842
lib/unistd/lseek.c 1842
lib/unistd/read.c 1843
lib/unistd/rmdir.c 1843
lib/unistd/seteuid.c 1844
lib/unistd/setpgrp.c 1844
lib/unistd/setuid.c 1844
lib/unistd/sleep.c 1845
lib/unistd/ttyname.c 1845
lib/unistd/unlink.c 1846
lib/unistd/write.c 1846
os16: «lib/utime.h» 1847
lib/utime/utime.c 1847

os16: file isolati della directory «lib/»

«

lib/NULL.h

«

Si veda la sezione u0.2.

```
2030001 #ifndef _NULL_H
2030002 #define _NULL_H      1
2030003
2030004 #define NULL 0
2030005
2030006 #endif
```

lib/SEEK.h

«

Si veda la sezione u0.2.

```
2040001 #ifndef _SEEK_H
2040002 #define _SEEK_H      1
2040003
2040004 //------------------------------------------------------------------------
2040005 // These values are used inside <stdio.h> and <unistd.h>
2040006 //------------------------------------------------------------------------
2040007 #define SEEK_SET        0  // From the start.
2040008 #define SEEK_CUR        1  // From current position.
2040009 #define SEEK_END        2  // From the end.
2040010 //------------------------------------------------------------------------
2040011
2040012 #endif
```

lib/_Bool.h

«

Si veda la sezione u0.2.

```
2050001 #ifndef __BOOL_H
2050002 #define __BOOL_H      1
2050003
2050004 typedef unsigned char _Bool;
2050005
2050006 #endif
```

lib/clock_t.h

«

Si veda la sezione u0.2.

```
2060001 #ifndef _CLOCK_T_H
2060002 #define _CLOCK_T_H       1
2060003
2060004 typedef unsigned long int clock_t;      // 32 bit unsigned int.
2060005
2060006 #endif
```

lib/const.h

«

Si veda la sezione u0.2.

```
2070001 #ifndef _CONST_H
2070002 #define _CONST_H        1
2070003
2070004 #define const
2070005
2070006 #endif
```

lib/ctype.h

«

Si veda la sezione u0.2.

```
2080001 #ifndef _CTYPE_H
2080002 #define _CTYPE_H       1
2080003 //---------------------------------------------------------------------------
2080004
2080005 #include <NULL.h>
2080006 //---------------------------------------------------------------------------
2080007 #define isblank(C)  ((int) (C == ' ' || C == '\t'))
2080008 #define isspace(C)  ((int) (C == ' ' \
2080009                             || C == '\f' \
2080010                             || C == '\n' \
2080011                             || C == '\n' \
2080012                             || C == '\r' \
2080013                             || C == '\t' \
2080014                             || C == '\v'))
2080015 #define isdigit(C)  ((int) (C >= '0' && C <= '9'))
2080016 #define isxdigit(C) ((int) ((C >= '0' && C <= '9') \
2080017                             || (C >= 'A' && C <= 'F') \
2080018                             || (C >= 'a' && C <= 'f')))
2080019 #define isupper(C)  ((int) (C >= 'A' && C <= 'Z'))
2080020 #define islower(C)  ((int) (C >= 'a' && C <= 'z'))
2080021 #define iscntrl(C)  ((int) ((C >= 0x00 && C <= 0x1F) || C == 0x7F))
2080022 #define isgraph(C)  ((int) (C >= 0x21 && C <= 0x7E))
2080023 #define isprint(C)  ((int) (C >= 0x20 && C <= 0x7E))
2080024 #define isalpha(C)  (isupper (C) || islower (C))
2080025 #define isalnum(C)  (isalpha (C) || isdigit (C))
2080026 #define ispunct(C)  (isgraph (C) && (!isspace (C)) && (!isalnum (C)))
2080027 #define tolower(C)  (isupper (C) ? ((C) + 0x20) : (C))
2080028 #define toupper(C)  (islower (C) ? ((C) - 0x20) : (C))
2080029 #define toascii(C)  (C & 0x7F)
2080030 #define _tolower(C) (isupper (C) ? ((C) + 0x20) : (C))
2080031 #define _toupper(C) (islower (C) ? ((C) - 0x20) : (C))
2080032 //---------------------------------------------------------------------------
2080033
2080034 #endif
```

lib/inttypes.h

«

Si veda la sezione u0.2.

```
2090001 #ifndef _INTTYPES_H
2090002 #define _INTTYPES_H     1
2090003 //---------------------------------------------------------------------------
2090004
2090005 #include <const.h>
2090006 #include <restrict.h>
2090007 #include <stdint.h>
2090008 #include <wchar_t.h>
2090009 //---------------------------------------------------------------------------
2090010 typedef struct {
2090011     intmax_t quot;
2090012     intmax_t rem;
2090013 } imaxdiv_t;
2090014 //
2090015 imaxdiv_t imaxdiv    (intmax_t numer, intmax_t denom);
2090016 //---------------------------------------------------------------------------
2090017 // Output typesetting.
2090018 //---------------------------------------------------------------------------
2090019 #define PRId8           "d"
2090020 #define PRId16          "d"
2090021 #define PRId32          "ld"
2090022 #define PRIdLEAST8      "d"
2090023 #define PRIdLEAST16     "d"
2090024 #define PRIdLEAST32     "ld"
2090025 #define PRIdFAST8       "d"
2090026 #define PRIdFAST16      "d"
2090027 #define PRIdFAST32      "ld"
2090028 #define PRIdMAX         "ld"
2090029 #define PRIdPTR         "d"
2090030 #define PRIi8           "i"
2090031 #define PRIi16          "i"
2090032 #define PRIi32          "li"
2090033 #define PRIiLEAST8      "i"
2090034 #define PRIiLEAST16     "i"
```



```
2090035 #define PRIiLEAST32     "li"
2090036 #define PRIiFAST8       "i"
2090037 #define PRIiFAST16      "i"
2090038 #define PRIiFAST32      "i"
2090039 #define PRIiMAX         "li"
2090040 #define PRIiPTR         "i"
2090041 #define PRIo8           "o"
2090042 #define PRIo16          "o"
2090043 #define PRIo32          "lo"
2090044 #define PRIoLEAST8      "o"
2090045 #define PRIoLEAST16     "o"
2090046 #define PRIoLEAST32     "lo"
2090047 #define PRIoFAST8       "o"
2090048 #define PRIoFAST16      "o"
2090049 #define PRIoFAST32      "lo"
2090050 #define PRIoMAX         "lo"
2090051 #define PRIoPTR         "o"
2090052 #define PRIu8           "u"
2090053 #define PRIu16          "u"
2090054 #define PRIu32          "lu"
2090055 #define PRIuLEAST8      "u"
2090056 #define PRIuLEAST16     "u"
2090057 #define PRIuLEAST32     "lu"
2090058 #define PRIuFAST8       "u"
2090059 #define PRIuFAST16      "u"
2090060 #define PRIuFAST32      "lu"
2090061 #define PRIuMAX         "lu"
2090062 #define PRIuPTR         "u"
2090063 #define PRIx8           "x"
2090064 #define PRIx16          "x"
2090065 #define PRIx32          "lx"
2090066 #define PRIxLEAST8      "x"
2090067 #define PRIxLEAST16     "x"
2090068 #define PRIxLEAST32     "lx"
2090069 #define PRIxFAST8       "x"
2090070 #define PRIxFAST16      "x"
2090071 #define PRIxFAST32      "lx"
2090072 #define PRIxMAX         "lx"
2090073 #define PRIxPTR         "x"
2090074 #define PRIX8           "X"
2090075 #define PRIX16          "X"
2090076 #define PRIX32          "lX"
2090077 #define PRIXLEAST8      "X"
2090078 #define PRIXLEAST16     "X"
2090079 #define PRIXLEAST32     "lX"
2090080 #define PRIXFAST8       "X"
2090081 #define PRIXFAST16      "X"
2090082 #define PRIXFAST32      "lX"
2090083 #define PRIXMAX         "lX"
2090084 #define PRIXPTR         "X"
2090085 //---------------------------------------------------------------------------
2090086 // Input scan and evaluation.
2090087 //---------------------------------------------------------------------------
2090088 #define SCNd8           "hhd"
2090089 #define SCNd16          "hd"
2090090 #define SCNd32          "d"
2090091 #define SCNdLEAST8      "hhd"
2090092 #define SCNdLEAST16     "hd"
2090093 #define SCNdLEAST32     "d"
2090094 #define SCNdFAST8       "hhd"
2090095 #define SCNdFAST16      "d"
2090096 #define SCNdFAST32      "d"
2090097 #define SCNdMAX         "ld"
2090098 #define SCNdPTR         "d"
2090099 #define SCNi8           "hhi"
2090100 #define SCNi16          "hi"
2090101 #define SCNi32          "i"
2090102 #define SCNiLEAST8      "hhi"
2090103 #define SCNiLEAST16     "hi"
2090104 #define SCNiLEAST32     "i"
2090105 #define SCNiFAST8       "hhi"
2090106 #define SCNiFAST16      "i"
2090107 #define SCNiFAST32      "i"
2090108 #define SCNiMAX         "li"
2090109 #define SCNiPTR         "i"
2090110 #define SCNo8           "hho"
2090111 #define SCNo16          "ho"
2090112 #define SCNo32          "o"
2090113 #define SCNoLEAST8      "hho"
2090114 #define SCNoLEAST16     "ho"
2090115 #define SCNoLEAST32     "o"
2090116 #define SCNoFAST8       "hho"
2090117 #define SCNoFAST16      "o"
2090118 #define SCNoFAST32      "o"
2090119 #define SCNoMAX         "lo"
2090120 #define SCNoPTR         "o"
2090121 #define SCNu8           "hhu"
2090122 #define SCNu16          "hu"
2090123 #define SCNu32          "u"
2090124 #define SCNuLEAST8      "hhu"
2090125 #define SCNuLEAST16     "hu"
2090126 #define SCNuLEAST32     "u"
2090127 #define SCNuFAST8       "hhu"
2090128 #define SCNuFAST16      "u"
2090129 #define SCNuFAST32      "u"
2090130 #define SCNuMAX         "lu"
2090131 #define SCNuPTR         "u"
2090132 #define SCNx8           "hhx"
2090133 #define SCNx16          "hx"
2090134 #define SCNx32          "x"
2090135 #define SCNxLEAST8      "hhx"
```

```
#define SCNxLEAST16     "hx"
#define SCNxLEAST32     "x"
#define SCNxFAST8       "hhx"
#define SCNxFAST16      "x"
#define SCNxFAST32      "x"
#define SCNxMAX         "lx"
#define SCNxPTR         "x"
//------------------------------------------------------------------------
intmax_t  strtoimax  (const char *restrict nptr,
                      char **restrict endptr, int base);
uintmax_t strtouimax (const char *restrict nptr,
                      char **restrict endptr, int base);
intmax_t  wcstoimax  (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr, int base);
uintmax_t wcstouimax (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr, int base);
//------------------------------------------------------------------------

#endif
```

lib/limits.h

Si veda la sezione u0.2.

```
#ifndef _LIMITS_H
#define _LIMITS_H       1
//------------------------------------------------------------------------
#define CHAR_BIT              (8)
#define SCHAR_MIN          (-0x80)
#define SCHAR_MAX           (0x7F)
#define UCHAR_MAX           (0xFF)
#define CHAR_MIN        SCHAR_MIN
#define CHAR_MAX        SCHAR_MAX
#define MB_LEN_MAX             (1)
#define SHRT_MIN         (-0x8000)
#define SHRT_MAX          (0x7FFF)
#define USHRT_MAX         (0xFFFF)
#define INT_MIN          (-0x8000)
#define INT_MAX           (0x7FFF)
#define UINT_MAX          (0xFFFFU)
#define LONG_MIN     (-0x80000000L)
#define LONG_MAX      (0x7FFFFFFFL)
#define ULONG_MAX     (0xFFFFFFFFUL)
//------------------------------------------------------------------------
#define LLONG_MIN    (-0x80000000L)     // The type 'long long int'
#define LLONG_MAX     (0x7FFFFFFFL)     // is not available with
#define ULLONG_MAX    (0xFFFFFFFFUL)    // a K&R C compiler.
//------------------------------------------------------------------------
#define WORD_BIT              16  // POSIX requires at least 32!
#define LONG_BIT              32
#define SSIZE_MAX        LONG_MAX
//------------------------------------------------------------------------
#define ARG_MAX             1024  // Arguments+environment max length.
#define ATEXIT_MAX            32  // Max "at exit" functions.
#define FILESIZEBITS          32  // File size needs integer size...
#define LINK_MAX             254  // Max links per file.
#define NAME_MAX              14  // File name max (Minix 1 fs).
#define OPEN_MAX               8  // Max open files per process.
#define PATH_MAX              64  // Path, including final '\0'.
#define MAX_CANON              1  // Max bytes in canonical tty queue.
#define MAX_INPUT              1  // Max bytes in tty input queue.
//------------------------------------------------------------------------
#define CHLD_MAX          INT_MAX  // Not used.
#define HOST_NAME_MAX     INT_MAX  // Not used.
#define LOGIN_NAME_MAX    INT_MAX  // Not used.
#define PAGE_SIZE         INT_MAX  // Not used.
#define RE_DUP_MAX        INT_MAX  // Not used.
#define STREAM_MAX        INT_MAX  // Not used.
#define SYMLOOP_MAX       INT_MAX  // Not used.
#define TTY_NAME_MAX      INT_MAX  // Not used.
#define TZNAME_MAX        INT_MAX  // Not used.
#define PIPE_MAX          INT_MAX  // Not used.
#define SYMLINK_MAX       INT_MAX  // Not used.
//------------------------------------------------------------------------

#endif
```

lib/ptrdiff_t.h

Si veda la sezione u0.2.

```
#ifndef _PTRDIFF_T_H
#define _PTRDIFF_T_H    1

typedef int ptrdiff_t;

#endif
```

lib/restrict.h

Si veda la sezione u0.2.

```
#ifndef _RESTRICT_H
#define _RESTRICT_H     1

#define restrict

#endif
```


lib/size_t.h

Si veda la sezione u0.2.

```
#ifndef _SIZE_T_H
#define _SIZE_T_H       1
//
// The type 'size_t' *must* be equal to an 'int'.
//
typedef unsigned int size_t;

#endif
```

lib/stdarg.h

Si veda la sezione u0.2.

```
#ifndef _STDARG_H
#define _STDARG_H       1
//------------------------------------------------------------------------
typedef unsigned char *va_list;
//------------------------------------------------------------------------

#define va_start(ap, last) ((void) ((ap) = \
                            ((va_list) &(last)) + (sizeof (last))))
#define va_end(ap)         ((void) ((ap) = 0))
#define va_copy(dest, src) ((void) ((dest) = (va_list) (src)))
#define va_arg(ap, type)   (((ap) = (ap) + (sizeof (type))), \
                            *((type *) ((ap) - (sizeof (type)))))
//------------------------------------------------------------------------

#endif
```

lib/stdbool.h

Si veda la sezione u0.2.

```
#ifndef _STDBOOL_H
#define _STDBOOL_H      1
//------------------------------------------------------------------------
typedef unsigned char bool;     // [1]
#define true   ((bool) 1)
#define false  ((bool) 0)
#define __bool_true_false_are_defined   1
//
// [1] For some reason, it cannot be defined as a macro expanding to
//     '_Bool'. Anyway, it is the same kind of type.
//
//------------------------------------------------------------------------

#endif
```

lib/stddef.h

Si veda la sezione u0.2.

```
#ifndef _STDDEF_H
#define _STDDEF_H      1
//------------------------------------------------------------------------

#include <ptrdiff_t.h>
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>
//------------------------------------------------------------------------
#define offsetof(type, member)  ((size_t) &((type *)0)->member)
//------------------------------------------------------------------------

#endif
```

lib/stdint.h

Si veda la sezione u0.2.

```
#ifndef _STDINT_H
#define _STDINT_H       1
//------------------------------------------------------------------------
typedef signed char                  int8_t;
typedef short int                   int16_t;
typedef long int                    int32_t;
typedef unsigned char               uint8_t;
typedef unsigned short int         uint16_t;
typedef unsigned long int          uint32_t;
//
#define INT8_MIN                     (-0x80)
#define INT8_MAX                      (0x7F)
#define UINT8_MAX                     (0xFF)
#define INT16_MIN                  (-0x8000)
#define INT16_MAX                   (0x7FFF)
#define UINT16_MAX                  (0xFFFF)
#define INT32_MIN              (-0x80000000)
#define INT32_MAX               (0x7FFFFFFF)
#define UINT32_MAX              (0xFFFFFFFFU)
//------------------------------------------------------------------------
typedef signed char             int_least8_t;
typedef short int              int_least16_t;
typedef long int               int_least32_t;
typedef unsigned char          uint_least8_t;
```

```
2170025  typedef unsigned short int    uint_least16_t;
2170026  typedef unsigned long int     uint_least32_t;
2170027  //
2170028  #define INT_LEAST8_MIN               (-0x80)
2170029  #define INT_LEAST8_MAX                (0x7F)
2170030  #define UINT_LEAST8_MAX               (0xFF)
2170031  #define INT_LEAST16_MIN            (-0x8000)
2170032  #define INT_LEAST16_MAX             (0x7FFF)
2170033  #define UINT_LEAST16_MAX            (0xFFFF)
2170034  #define INT_LEAST32_MIN         (-0x80000000)
2170035  #define INT_LEAST32_MAX          (0x7FFFFFFF)
2170036  #define UINT_LEAST32_MAX        (0xFFFFFFFFU)
2170037  //------------------------------------------------------------------------
2170038  #define INT8_C(VAL)              VAL
2170039  #define INT16_C(VAL)             VAL
2170040  #define INT32_C(VAL)             VAL
2170041  #define UINT8_C(VAL)             VAL
2170042  #define UINT16_C(VAL)            VAL
2170043  #define UINT32_C(VAL)            VAL ## U
2170044  //------------------------------------------------------------------------
2170045  typedef signed char              int_fast8_t;
2170046  typedef int                     int_fast16_t;
2170047  typedef long int                int_fast32_t;
2170048  typedef unsigned char           uint_fast8_t;
2170049  typedef unsigned int           uint_fast16_t;
2170050  typedef unsigned long int      uint_fast32_t;
2170051  //
2170052  #define INT_FAST8_MIN                (-0x80)
2170053  #define INT_FAST8_MAX                 (0x7F)
2170054  #define UINT_FAST8_MAX                (0xFF)
2170055  #define INT_FAST16_MIN          (-0x80000000)
2170056  #define INT_FAST16_MAX           (0x7FFFFFFF)
2170057  #define UINT_FAST16_MAX          (0xFFFFFFFFU)
2170058  #define INT_FAST32_MIN          (-0x80000000)
2170059  #define INT_FAST32_MAX           (0x7FFFFFFF)
2170060  #define UINT_FAST32_MAX          (0xFFFFFFFFU)
2170061  //------------------------------------------------------------------------
2170062  typedef int                         intptr_t;
2170063  typedef unsigned int               uintptr_t;
2170064  //
2170065  #define INTPTR_MIN              (-0x80000000)
2170066  #define INTPTR_MAX              (0x7FFFFFFF)
2170067  #define UINTPTR_MAX             (0xFFFFFFFFU)
2170068  //------------------------------------------------------------------------
2170069  typedef long int                   intmax_t;
2170070  typedef unsigned long int         uintmax_t;
2170071  //
2170072  #define INTMAX_C(VAL)             VAL ## L
2170073  #define UINTMAX_C(VAL)            VAL ## UL
2170074  //
2170075  #define INTMAX_MIN             (-0x80000000L)
2170076  #define INTMAX_MAX              (0x7FFFFFFFL)
2170077  #define UINTMAX_MAX             (0xFFFFFFFFUL)
2170078  //------------------------------------------------------------------------
2170079  #define PTRDIFF_MIN            (-0x80000000)
2170080  #define PTRDIFF_MAX             (0x7FFFFFFF)
2170081  //
2170082  #define SIG_ATOMIC_MIN         (-0x80000000)
2170083  #define SIG_ATOMIC_MAX          (0x7FFFFFFF)
2170084  //
2170085  #define SIZE_MAX                    (0xFFFFU)
2170086  //
2170087  #define WCHAR_MIN                      (0)
2170088  #define WCHAR_MAX                   (0xFFU)
2170089  //
2170090  #define WINT_MIN                    (-0x80L)
2170091  #define WINT_MAX                     (0x7FL)
2170092  //------------------------------------------------------------------------
2170093
2170094  #endif
```

lib/time_t.h

«

Si veda la sezione u0.2.

```
2180001  #ifndef _TIME_T_H
2180002  #define _TIME_T_H       1
2180003
2180004  typedef long int time_t;
2180005
2180006  #endif
```

lib/wchar_t.h

«

Si veda la sezione u0.2.

```
2190001  #ifndef _WCHAR_T_H
2190002  #define _WCHAR_T_H      1
2190003
2190004  typedef unsigned char wchar_t;
2190005
2190006  #endif
```


os16: «lib/dirent.h»

«

Si veda la sezione u0.2.

```
2200001  #ifndef _DIRENT_H
2200002  #define _DIRENT_H        1
2200003
2200004  #include <sys/types.h>  // ino_t
2200005  #include <limits.h>     // NAME_MAX
2200006  #include <const.h>
2200007
2200008  //------------------------------------------------------------------------
2200009  struct dirent {
2200010      ino_t  d_ino;                  // I-node number [1]
2200011      char   d_name[NAME_MAX+1];     // NAME_MAX + Null termination
2200012  };
2200013  //
2200014  // [1] The type 'ino_t' must be equal to 'uint16_t', because the
2200015  //     directory inside the Minix 1 file system has exactly such
2200016  //     size.
2200017  //
2200018  //------------------------------------------------------------------------
2200019  #define DOPEN_MAX   OPEN_MAX/2  // <limits.h> [1]
2200020  //
2200021  // [1] DOPEN_MAX is not standard, but it is used to define how many
2200022  //     directory slot to keep for open directories. As directory streams
2200023  //     are opened as file descriptors, the sum of all kind of file open
2200024  //     cannot be more than OPEM_MAX.
2200025  //------------------------------------------------------------------------
2200026  typedef struct {
2200027      int           fdn;       // File descriptor number.
2200028      struct dirent dir;       // Last directory item read.
2200029  } DIR;
2200030
2200031  extern DIR _directory_stream[]; // Defined inside 'lib/dirent/DIR.c'.
2200032  //------------------------------------------------------------------------
2200033  // Function prototypes.
2200034  //------------------------------------------------------------------------
2200035  int            closedir  (DIR *dp);
2200036  DIR           *opendir   (const char *name);
2200037  struct dirent *readdir   (DIR *dp);
2200038  void           rewinddir (DIR *dp);
2200039  //------------------------------------------------------------------------
2200040
2200041  #endif
```

lib/dirent/DIR.c

«

Si veda la sezione u0.2.

```
2210001  #include <dirent.h>
2210002  //
2210003  // There must be room for at least 'DOPEN_MAX' elements.
2210004  //
2210005  DIR _directory_stream[DOPEN_MAX];
2210006
2210007  void
2210008  _dirent_directory_stream_setup (void)
2210009  {
2210010      int d;
2210011      //
2210012      for (d = 0; d < DOPEN_MAX; d++)
2210013        {
2210014          _directory_stream[d].fdn   = -1;
2210015        }
2210016  }
```

lib/dirent/closedir.c

«

Si veda la sezione u0.10.

```
2220001  #include <dirent.h>
2220002  #include <fcntl.h>
2220003  #include <const.h>
2220004  #include <sys/types.h>
2220005  #include <sys/stat.h>
2220006  #include <unistd.h>
2220007  #include <errno.h>
2220008  #include <stddef.h>
2220009  //------------------------------------------------------------------------
2220010  int
2220011  closedir (DIR *dp)
2220012  {
2220013      //
2220014      // Check for a valid argument
2220015      //
2220016      if (dp == NULL)
2220017        {
2220018          //
2220019          // Not a valid pointer.
2220020          //
2220021          errset (EBADF);         // Invalid directory.
2220022          return (-1);
2220023        }
2220024      //
2220025      // Check if it is an open directory stream.
2220026      //
2220027      if (dp->fdn < 0)
2220028        {
2220029          //
```

```
2220030         // The stream is closed.
2220031         //
2220032         errset (EBADF);         // Invalid directory.
2220033         return (-1);
2220034       }
2220035     //
2220036     // Close the file descriptor. It there is an error,
2220037     // the 'errno' variable will be set by 'close()'.
2220038     //
2220039     return (close (dp->fdn));
2220040 }
```

lib/dirent/opendir.c

« Si veda la sezione u0.76.

```
2230001 #include <dirent.h>
2230002 #include <fcntl.h>
2230003 #include <const.h>
2230004 #include <sys/types.h>
2230005 #include <sys/stat.h>
2230006 #include <unistd.h>
2230007 #include <errno.h>
2230008 #include <stddef.h>
2230009 //---------------------------------------------------------------------
2230010 DIR *
2230011 opendir (const char *path)
2230012 {
2230013     int   fdn;
2230014     int   d;
2230015     DIR  *dp;
2230016     struct stat file_status;
2230017     //
2230018     // Function 'opendir()' is used only for reading.
2230019     //
2230020     fdn = open (path, O_RDONLY);
2230021     //
2230022     // Check the file descriptor returned.
2230023     //
2230024     if (fdn < 0)
2230025       {
2230026         //
2230027         // The variable 'errno' is already set:
2230028         //   EINVAL
2230029         //   EMFILE
2230030         //   ENFILE
2230031         //
2230032         errset (errno);
2230033         return (NULL);
2230034       }
2230035     //
2230036     // Set the 'FD_CLOEXEC' flag for that file descriptor.
2230037     //
2230038     if (fcntl (fdn, F_SETFD, FD_CLOEXEC) != 0)
2230039       {
2230040         //
2230041         // The variable 'errno' is already set:
2230042         //   EBADF
2230043         //
2230044         errset (errno);
2230045         close (fdn);
2230046         return (NULL);
2230047       }
2230048     //
2230049     //
2230050     //
2230051     if (fstat (fdn, &file_status) != 0)
2230052       {
2230053         //
2230054         // Error should be already set.
2230055         //
2230056         errset (errno);
2230057         close (fdn);
2230058         return (NULL);
2230059       }
2230060     //
2230061     // Verify it is a directory
2230062     //
2230063     if (!S_ISDIR(file_status.st_mode))
2230064       {
2230065         //
2230066         // It is not a directory!
2230067         //
2230068         close (fdn);
2230069         errset (ENOTDIR);               // Is not a directory.
2230070         return (NULL);
2230071       }
2230072     //
2230073     // A valid file descriptor is available: must find a free
2230074     // '_directory_stream[]' slot.
2230075     //
2230076     for (d = 0; d < DOPEN_MAX; d++)
2230077       {
2230078         if (_directory_stream[d].fdn < 0)
2230079           {
2230080             //
2230081             // Found a free slot: set it up.
2230082             //
2230083             dp = &(_directory_stream[d]);
2230084             dp->fdn   = fdn;
2230085             //
```



```
2230086             // Return the directory pointer.
2230087             //
2230088             return (dp);
2230089           }
2230090       }
2230091     //
2230092     // If we are here, there was no free directory slot available.
2230093     //
2230094     close  (fdn);
2230095     errset (EMFILE);                    // Too many file open.
2230096     return (NULL);
2230097 }
```

lib/dirent/readdir.c

« Si veda la sezione u0.86.

```
2240001 #include <dirent.h>
2240002 #include <fcntl.h>
2240003 #include <sys/types.h>
2240004 #include <sys/stat.h>
2240005 #include <unistd.h>
2240006 #include <errno.h>
2240007 #include <stddef.h>
2240008 //---------------------------------------------------------------------
2240009 struct dirent *
2240010 readdir (DIR *dp)
2240011 {
2240012     ssize_t size;
2240013     //
2240014     // Check for a valid argument.
2240015     //
2240016     if (dp == NULL)
2240017       {
2240018         //
2240019         // Not a valid pointer.
2240020         //
2240021         errset (EBADF);         // Invalid directory.
2240022         return (NULL);
2240023       }
2240024     //
2240025     // Check if it is an open directory stream.
2240026     //
2240027     if (dp->fdn < 0)
2240028       {
2240029         //
2240030         // The stream is closed.
2240031         //
2240032         errset (EBADF);         // Invalid directory.
2240033         return (NULL);
2240034       }
2240035     //
2240036     // Read the directory.
2240037     //
2240038     size = read (dp->fdn, &(dp->dir),
2240039                  (size_t) 16);
2240040     //
2240041     // Fix the null termination, if the name is very long.
2240042     //
2240043     dp->dir.d_name[NAME_MAX] = '\0';
2240044     //
2240045     // Check what was read.
2240046     //
2240047     if (size == 0)
2240048       {
2240049         //
2240050         // End of directory, but it is not an error.
2240051         //
2240052         return (NULL);
2240053       }
2240054     //
2240055     if (size < 0)
2240056       {
2240057         //
2240058         // This is an error. The variable 'errno' is already set.
2240059         //
2240060         errset (errno);
2240061         return (NULL);
2240062       }
2240063     //
2240064     if (dp->dir.d_ino == 0)
2240065       {
2240066         //
2240067         // This is a null directory record.
2240068         // Should try to read the next one.
2240069         //
2240070         return (readdir (dp));
2240071       }
2240072     //
2240073     if (strlen (dp->dir.d_name) == 0)
2240074       {
2240075         //
2240076         // This is a bad directory record: try to read next.
2240077         //
2240078         return (readdir (dp));
2240079       }
2240080     //
2240081     // A valid directory record should be available now.
2240082     //
2240083     return (&(dp->dir));
2240084 }
```

lib/dirent/rewinddir.c

«

Si veda la sezione u0.89.

```
2250001 #include <dirent.h>
2250002 #include <fcntl.h>
2250003 #include <const.h>
2250004 #include <sys/types.h>
2250005 #include <sys/stat.h>
2250006 #include <unistd.h>
2250007 #include <errno.h>
2250008 #include <stddef.h>
2250009 #include <stdio.h>
2250010 //-----------------------------------------------------------------------
2250011 void
2250012 rewinddir (DIR *dp)
2250013 {
2250014     FILE *fp;
2250015     //
2250016     // Check for a valid argument.
2250017     //
2250018     if (dp == NULL)
2250019       {
2250020         //
2250021         // Nothing to rewind, and no error to set.
2250022         //
2250023         return;
2250024       }
2250025     //
2250026     // Check if it is an open directory stream.
2250027     //
2250028     if (dp->fdn < 0)
2250029       {
2250030         //
2250031         // The stream is closed.
2250032         // Nothing to rewind, and no error to set.
2250033         //
2250034         return;
2250035       }
2250036     //
2250037     //
2250038     //
2250039     fp = &_stream[dp->fdn];
2250040     //
2250041     rewind (fp);
2250042 }
```

os16: «lib/errno.h»

«

Si veda la sezione u0.18.

```
2260001 #ifndef _ERRNO_H
2260002 #define _ERRNO_H       1
2260003
2260004 #include <limits.h>
2260005 #include <string.h>
2260006 //-----------------------------------------------------------------------
2260007 // The variable 'errno' is standard, but 'errln' and 'errfn' are added
2260008 // to keep track of the error source. Variable 'errln' is used to save
2260009 // the source file line number; variable 'errfn' is used to save the
2260010 // source file name. To set these variable in a consistent way it is
2260011 // also added a macro-instruction: 'errset'.
2260012 //-----------------------------------------------------------------------
2260013 extern int   errno;
2260014 extern int   errln;
2260015 extern char  errfn[PATH_MAX];
2260016 #define errset(e)       (errln = __LINE__, \
2260017                          strncpy (errfn, __FILE__, PATH_MAX), \
2260018                          errno = e)
2260019 //-----------------------------------------------------------------------
2260020 // Standard POSIX 'errno' macro variables.
2260021 //-----------------------------------------------------------------------
2260022 #define E2BIG            1 // Argument list too long.
2260023 #define EACCES           2 // Permission denied.
2260024 #define EADDRINUSE       3 // Address in use.
2260025 #define EADDRNOTAVAIL    4 // Address not available.
2260026 #define EAFNOSUPPORT     5 // Address family not supported.
2260027 #define EAGAIN           6 // Resource unavailable, try again.
2260028 #define EALREADY         7 // Connection already in progress.
2260029 #define EBADF            8 // Bad file descriptor.
2260030 #define EBADMSG          9 // Bad message.
2260031 #define EBUSY           10 // Device or resource busy.
2260032 #define ECANCELED       11 // Operation canceled.
2260033 #define ECHILD          12 // No child processes.
2260034 #define ECONNABORTED    13 // Connection aborted.
2260035 #define ECONNREFUSED    14 // Connection refused.
2260036 #define ECONNRESET      15 // Connection reset.
2260037 #define EDEADLK         16 // Resource deadlock would occur.
2260038 #define EDESTADDRREQ    17 // Destination address required.
2260039 #define EDOM            18 // Mathematics argument out of domain of
2260040                            // function.
2260041 #define EDQUOT          19 // Reserved.
2260042 #define EEXIST          20 // File exists.
2260043 #define EFAULT          21 // Bad address.
2260044 #define EFBIG           22 // File too large.
2260045 #define EHOSTUNREACH    23 // Host is unreachable.
2260046 #define EIDRM           24 // Identifier removed.
2260047 #define EILSEQ          25 // Illegal byte sequence.
2260048 #define EINPROGRESS     26 // Operation in progress.
2260049 #define EINTR           27 // Interrupted function.
2260050 #define EINVAL          28 // Invalid argument.
```



```
2260051 #define EIO             29 // I/O error.
2260052 #define EISCONN         30 // Socket is connected.
2260053 #define EISDIR          31 // Is a directory.
2260054 #define ELOOP           32 // Too many levels of symbolic links.
2260055 #define EMFILE          33 // Too many open files.
2260056 #define EMLINK          34 // Too many links.
2260057 #define EMSGSIZE        35 // Message too large.
2260058 #define EMULTIHOP       36 // Reserved.
2260059 #define ENAMETOOLONG    37 // Filename too long.
2260060 #define ENETDOWN        38 // Network is down.
2260061 #define ENETRESET       39 // Connection aborted by network.
2260062 #define ENETUNREACH     40 // Network unreachable.
2260063 #define ENFILE          41 // Too many files open in system.
2260064 #define ENOBUFS         42 // No buffer space available.
2260065 #define ENODATA         43 // No message is available on the stream head
2260066                            // read queue.
2260067 #define ENODEV          44 // No such device.
2260068 #define ENOENT          45 // No such file or directory.
2260069 #define ENOEXEC         46 // Executable file format error.
2260070 #define ENOLCK          47 // No locks available.
2260071 #define ENOLINK         48 // Reserved.
2260072 #define ENOMEM          49 // Not enough space.
2260073 #define ENOMSG          50 // No message of the desired type.
2260074 #define ENOPROTOOPT     51 // Protocol not available.
2260075 #define ENOSPC          52 // No space left on device.
2260076 #define ENOSR           53 // No stream resources.
2260077 #define ENOSTR          54 // Not a stream.
2260078 #define ENOSYS          55 // Function not supported.
2260079 #define ENOTCONN        56 // The socket is not connected.
2260080 #define ENOTDIR         57 // Not a directory.
2260081 #define ENOTEMPTY       58 // Directory not empty.
2260082 #define ENOTSOCK        59 // Not a socket.
2260083 #define ENOTSUP         60 // Not supported.
2260084 #define ENOTTY          61 // Inappropriate I/O control operation.
2260085 #define ENXIO           62 // No such device or address.
2260086 #define EOPNOTSUPP      63 // Operation not supported on socket.
2260087 #define EOVERFLOW       64 // Value too large to be stored in data type.
2260088 #define EPERM           65 // Operation not permitted.
2260089 #define EPIPE           66 // Broken pipe.
2260090 #define EPROTO          67 // Protocol error.
2260091 #define EPROTONOSUPPORT 68 // Protocol not supported.
2260092 #define EPROTOTYPE      69 // Protocol wrong type for socket.
2260093 #define ERANGE          70 // Result too large.
2260094 #define EROFS           71 // Read-only file system.
2260095 #define ESPIPE          72 // Invalid seek.
2260096 #define ESRCH           73 // No such process.
2260097 #define ESTALE          74 // Reserved.
2260098 #define ETIME           75 // Stream ioctl() timeout.
2260099 #define ETIMEDOUT       76 // Connection timed out.
2260100 #define ETXTBSY         77 // Text file busy.
2260101 #define EWOULDBLOCK     78 // Operation would block
2260102                            // (may be the same as EAGAIN).
2260103 #define EXDEV           79 // Cross-device link.
2260104 //-----------------------------------------------------------------------
2260105 // Added os16 errors.
2260106 //-----------------------------------------------------------------------
2260107 #define EUNKNOWN                  (-1) // Unknown error.
2260108 #define E_FILE_TYPE                80  // File type not compatible.
2260109 #define E_ROOT_INODE_NOT_CACHED    81  // The root directory inode is
2260110                                        // not cached.
2260111 #define E_CANNOT_READ_SUPERBLOCK   83  // Cannot read super block.
2260112 #define E_MAP_INODE_TOO_BIG        84  // Map inode too big.
2260113 #define E_MAP_ZONE_TOO_BIG         85  // Map zone too big.
2260114 #define E_DATA_ZONE_TOO_BIG        86  // Data zone too big.
2260115 #define E_CANNOT_FIND_ROOT_DEVICE  87  // Cannot find root device.
2260116 #define E_CANNOT_FIND_ROOT_INODE   88  // Cannot find root inode.
2260117 #define E_FILE_TYPE_UNSUPPORTED    89  // File type unsupported.
2260118 #define E_ENV_TOO_BIG              90  // Environment too big.
2260119 #define E_LIMIT                    91  // Exceeded implementation
2260120                                        // limits.
2260121 #define E_NOT_MOUNTED              92  // Not mounted.
2260122 #define E_NOT_IMPLEMENTED          93  // Not implemented.
2260123 //-----------------------------------------------------------------------
2260124 // Default descriptions for errors.
2260125 //-----------------------------------------------------------------------
2260126 #define TEXT_E2BIG                    "Argument list too long."
2260127 #define TEXT_EACCES                   "Permission denied."
2260128 #define TEXT_EADDRINUSE               "Address in use."
2260129 #define TEXT_EADDRNOTAVAIL            "Address not available."
2260130 #define TEXT_EAFNOSUPPORT             "Address family not supported."
2260131 #define TEXT_EAGAIN                   "Resource unavailable, " \
2260132                                       "try again."
2260133 #define TEXT_EALREADY                 "Connection already in " \
2260134                                       "progress."
2260135 #define TEXT_EBADF                    "Bad file descriptor."
2260136 #define TEXT_EBADMSG                  "Bad message."
2260137 #define TEXT_EBUSY                    "Device or resource busy."
2260138 #define TEXT_ECANCELED                "Operation canceled."
2260139 #define TEXT_ECHILD                   "No child processes."
2260140 #define TEXT_ECONNABORTED             "Connection aborted."
2260141 #define TEXT_ECONNREFUSED             "Connection refused."
2260142 #define TEXT_ECONNRESET               "Connection reset."
2260143 #define TEXT_EDEADLK                  "Resource deadlock would occur."
2260144 #define TEXT_EDESTADDRREQ             "Destination address required."
2260145 #define TEXT_EDOM                     "Mathematics argument out of " \
2260146                                       "domain of function."
2260147 #define TEXT_EDQUOT                   "Reserved error: EDQUOT"
2260148 #define TEXT_EEXIST                   "File exists."
2260149 #define TEXT_EFAULT                   "Bad address."
2260150 #define TEXT_EFBIG                    "File too large."
2260151 #define TEXT_EHOSTUNREACH             "Host is unreachable."
```

```
#define TEXT_EIDRM                      "Identifier removed."
#define TEXT_EILSEQ                     "Illegal byte sequence."
#define TEXT_EINPROGRESS                "Operation in progress."
#define TEXT_EINTR                      "Interrupted function."
#define TEXT_EINVAL                     "Invalid argument."
#define TEXT_EIO                        "I/O error."
#define TEXT_EISCONN                    "Socket is connected."
#define TEXT_EISDIR                     "Is a directory."
#define TEXT_ELOOP                      "Too many levels of " \
                                        "symbolic links."
#define TEXT_EMFILE                     "Too many open files."
#define TEXT_EMLINK                     "Too many links."
#define TEXT_EMSGSIZE                   "Message too large."
#define TEXT_EMULTIHOP                  "Reserved error: EMULTIHOP"
#define TEXT_ENAMETOOLONG               "Filename too long."
#define TEXT_ENETDOWN                   "Network is down."
#define TEXT_ENETRESET                  "Connection aborted by network."
#define TEXT_ENETUNREACH                "Network unreachable."
#define TEXT_ENFILE                     "Too many files open in system."
#define TEXT_ENOBUFS                    "No buffer space available."
#define TEXT_ENODATA                    "No message is available on " \
                                        "the stream head read queue."
#define TEXT_ENODEV                     "No such device."
#define TEXT_ENOENT                     "No such file or directory."
#define TEXT_ENOEXEC                    "Executable file format error."
#define TEXT_ENOLCK                     "No locks available."
#define TEXT_ENOLINK                    "Reserved error: ENOLINK"
#define TEXT_ENOMEM                     "Not enough space."
#define TEXT_ENOMSG                     "No message of the desired " \
                                        "type."
#define TEXT_ENOPROTOOPT                "Protocol not available."
#define TEXT_ENOSPC                     "No space left on device."
#define TEXT_ENOSR                      "No stream resources."
#define TEXT_ENOSTR                     "Not a stream."
#define TEXT_ENOSYS                     "Function not supported."
#define TEXT_ENOTCONN                   "The socket is not connected."
#define TEXT_ENOTDIR                    "Not a directory."
#define TEXT_ENOTEMPTY                  "Directory not empty."
#define TEXT_ENOTSOCK                   "Not a socket."
#define TEXT_ENOTSUP                    "Not supported."
#define TEXT_ENOTTY                     "Inappropriate I/O control " \
                                        "operation."
#define TEXT_ENXIO                      "No such device or address."
#define TEXT_EOPNOTSUPP                 "Operation not supported on " \
                                        "socket."
#define TEXT_EOVERFLOW                  "Value too large to be " \
                                        "stored in data type."
#define TEXT_EPERM                      "Operation not permitted."
#define TEXT_EPIPE                      "Broken pipe."
#define TEXT_EPROTO                     "Protocol error."
#define TEXT_EPROTONOSUPPORT            "Protocol not supported."
#define TEXT_EPROTOTYPE                 "Protocol wrong type for " \
                                        "socket."
#define TEXT_ERANGE                     "Result too large."
#define TEXT_EROFS                      "Read-only file system."
#define TEXT_ESPIPE                     "Invalid seek."
#define TEXT_ESRCH                      "No such process."
#define TEXT_ESTALE                     "Reserved error: ESTALE"
#define TEXT_ETIME                      "Stream ioctl() timeout."
#define TEXT_ETIMEDOUT                  "Connection timed out."
#define TEXT_ETXTBSY                    "Text file busy."
#define TEXT_EWOULDBLOCK                "Operation would block."
#define TEXT_EXDEV                      "Cross-device link."
//---------------------------------------------------------------------
#define TEXT_EUNKNOWN                   "Unknown error."
#define TEXT_E_FILE_TYPE                "File type not compatible."
#define TEXT_E_ROOT_INODE_NOT_CACHED    "The root directory inode " \
                                        "is not cached."
#define TEXT_E_CANNOT_READ_SUPERBLOCK   "Cannot read super block."
#define TEXT_E_MAP_INODE_TOO_BIG        "Map inode too big."
#define TEXT_E_MAP_ZONE_TOO_BIG         "Map zone too big."
#define TEXT_E_DATA_ZONE_TOO_BIG        "Data zone too big."
#define TEXT_E_CANNOT_FIND_ROOT_DEVICE  "Cannot find root device."
#define TEXT_E_CANNOT_FIND_ROOT_INODE   "Cannot find root inode."
#define TEXT_E_FILE_TYPE_UNSUPPORTED    "File type unsupported."
#define TEXT_E_ENV_TOO_BIG              "Environment too big."
#define TEXT_E_LIMIT                    "Exceeded implementation " \
                                        "limits."
#define TEXT_E_NOT_MOUNTED               "Not mounted."
#define TEXT_E_NOT_IMPLEMENTED          "Not implemented."

//---------------------------------------------------------------------
// The function 'error()' is not standard and is used to return a
// pointer to a string containing the default description of the
// error contained inside 'errno'.
//---------------------------------------------------------------------
char *error      (void);        // Not standard!

#endif
```

lib/errno/errno.c

«

Si veda la sezione u0.18.

```
//---------------------------------------------------------------------
// This file does not include the 'errno.h' header, because here 'errno'
// should not be declared as an extern variable!
//---------------------------------------------------------------------

#include <limits.h>
//---------------------------------------------------------------------
```



```
// The variable 'errno' is standard, but 'errln' and 'errfn' are added
// to keep track of the error source. Variable 'errln' is used to save
// the source file line number; variable 'errfn' is used to save the
// source file name. To set these variable in a consistent way it is
// also added a macro-instruction: 'errset'.
//---------------------------------------------------------------------
int  errno;
int  errln;
char errfn[PATH_MAX];
//---------------------------------------------------------------------
```

os16: «lib/fcntl.h»

«

Si veda la sezione u0.2.

```
#ifndef _FCNTL_H
#define _FCNTL_H        1

#include <const.h>
#include <sys/types.h>  // mode_t
                        // off_t
                        // pid_t
//---------------------------------------------------------------------
// Values for the second parameter of function 'fcntl()'.
//---------------------------------------------------------------------
#define F_DUPFD         0       // Duplicate file descriptor.
#define F_GETFD         1       // Get file descriptor flags.
#define F_SETFD         2       // Set file descriptor flags.
#define F_GETFL         3       // Get file status flags.
#define F_SETFL         4       // Set file status flags.
#define F_GETLK         5       // Get record locking information.
#define F_SETLK         6       // Set record locking information.
#define F_SETLKW        7       // Set record locking information;
                                // wait if blocked.
#define F_GETOWN        8       // Set owner of socket.
#define F_SETOWN        9       // Get owner of socket.
//---------------------------------------------------------------------
// Flags to be set with:
//   fcntl (fd, F_SETFD, ...);
//---------------------------------------------------------------------
#define FD_CLOEXEC      1       // Close the file descriptor upon
                                // execution of an exec() family
                                // function.
//---------------------------------------------------------------------
// Values for type 'l_type', used for record locking with 'fcntl()'.
//---------------------------------------------------------------------
#define F_RDLCK         0       // Read lock.
#define F_WRLCK         1       // Write lock.
#define F_UNLCK         2       // Remove lock.
//---------------------------------------------------------------------
// Flags for file creation, in place of 'oflag' parameter for function
// 'open()'.
//---------------------------------------------------------------------
#define O_CREAT         000010  // Create file if it does not exist.
#define O_EXCL          000020  // Exclusive use flag.
#define O_NOCTTY        000040  // Do not assign a controlling terminal.
#define O_TRUNC         000100  // Truncation flag.
//---------------------------------------------------------------------
// Flags for the file status, used with 'open()' and 'fcntl()'.
//---------------------------------------------------------------------
#define O_APPEND        000200  // Write append.
#define O_DSYNC         000400  // Synchronized write operations.
#define O_NONBLOCK      001000  // Non-blocking mode.
#define O_RSYNC         002000  // Synchronized read operations.
#define O_SYNC          004000  // Synchronized read and write.
//---------------------------------------------------------------------
// File access mask selection.
//---------------------------------------------------------------------
#define O_ACCMODE       000003  // Mask to select the last three bits,
                                // used to specify the main access
                                // modes: read, write and both.
//---------------------------------------------------------------------
// Main access modes.
//---------------------------------------------------------------------
#define O_RDONLY        000001                  // Read.
#define O_WRONLY        000002                  // Write.
#define O_RDWR          (O_RDONLY | O_WRONLY)   // Both read and write.
//---------------------------------------------------------------------
// Structure 'flock', used to file lock for POSIX standard. It is not
// used inside os16.
//---------------------------------------------------------------------
struct flock {
    short int l_type;   // Type of lock: F_RDLCK, F_WRLCK, or F_UNLCK.
    short int l_whence; // Start reference point.
    off_t     l_start;  // Offset, from 'l_whence', for the area start.
    off_t     l_len;    // Locked area size. Zero means up to the end of
                        // the file.
    pid_t     l_pid;    // The process id blocking the area.
};
//---------------------------------------------------------------------
// Function prototypes.
//---------------------------------------------------------------------
int creat (const char *path, mode_t mode);
int fcntl (int fdn, int cmd, ...);
int open  (const char *path, int oflags, ...);
//---------------------------------------------------------------------

#endif
```

lib/fcntl/creat.c

«

Si veda la sezione u0.11.

```
2290001 #include <fcntl.h>
2290002 #include <sys/types.h>
2290003 #include <const.h>
2290004 //-------------------------------------------------------------------------
2290005 int
2290006 creat (const char *path, mode_t mode)
2290007 {
2290008     return (open (path, O_WRONLY|O_CREAT|O_TRUNC, mode));
2290009 }
```

lib/fcntl/fcntl.c

«

Si veda la sezione u0.13.

```
2300001 #include <fcntl.h>
2300002 #include <stdarg.h>
2300003 #include <stddef.h>
2300004 #include <string.h>
2300005 #include <errno.h>
2300006 #include <sys/os16.h>
2300007 #include <const.h>
2300008 #include <limits.h>
2300009 //-------------------------------------------------------------------------
2300010 int
2300011 fcntl (int fdn, int cmd, ...)
2300012 {
2300013     va_list ap;
2300014     sysmsg_fcntl_t msg;
2300015     va_start (ap, cmd);
2300016     //
2300017     // Well known arguments.
2300018     //
2300019     msg.fdn = fdn;
2300020     msg.cmd = cmd;
2300021     //
2300022     // Select other arguments.
2300023     //
2300024     switch (cmd)
2300025       {
2300026         case F_DUPFD:
2300027         case F_SETFD:
2300028         case F_SETFL:
2300029             msg.arg = va_arg (ap, int);
2300030             break;
2300031         case F_GETFD:
2300032         case F_GETFL:
2300033             break;
2300034         case F_GETOWN:
2300035         case F_SETOWN:
2300036         case F_GETLK:
2300037         case F_SETLK:
2300038         case F_SETLKW:
2300039             errset (E_NOT_IMPLEMENTED);         // Not implemented.
2300040             return (-1);
2300041         default:
2300042             errset (EINVAL);                    // Not implemented.
2300043             return (NULL);
2300044       }
2300045     //
2300046     // Do the system call.
2300047     //
2300048     sys (SYS_FCNTL, &msg, (sizeof msg));
2300049     errno = msg.errno;
2300050     errln = msg.errln;
2300051     strncpy (errfn, msg.errfn, PATH_MAX);
2300052     return (msg.ret);
2300053 }
```

lib/fcntl/open.c

«

Si veda la sezione u0.28.

```
2310001 #include <fcntl.h>
2310002 #include <stdarg.h>
2310003 #include <stddef.h>
2310004 #include <string.h>
2310005 #include <errno.h>
2310006 #include <sys/os16.h>
2310007 #include <const.h>
2310008 #include <limits.h>
2310009 //-------------------------------------------------------------------------
2310010 int
2310011 open (const char *path, int oflags, ...)
2310012 {
2310013     va_list ap;
2310014     sysmsg_open_t msg;
2310015     va_start (ap, oflags);
2310016     if (path == NULL || strlen (path) == 0)
2310017       {
2310018         errset (EINVAL);        // Invalid argument.
2310019         return (-1);
2310020       }
2310021     strncpy (msg.path, path, PATH_MAX);
2310022     msg.flags = oflags;
2310023     msg.mode  = va_arg (ap, mode_t);
2310024     sys (SYS_OPEN, &msg, (sizeof msg));
2310025     errno = msg.errno;
2310026     errln = msg.errln;
2310027     strncpy (errfn, msg.errfn, PATH_MAX);
2310028     return (msg.ret);
2310029 }
```


os16: «lib/grp.h»

«

Si veda la sezione u0.2.

```
2320001 //-------------------------------------------------------------------------
2320002 // os16 does not have a group management!
2320003 //-------------------------------------------------------------------------
2320004
2320005 #ifndef _GRP_H
2320006 #define _GRP_H        1
2320007
2320008 #include <const.h>
2320009 #include <restrict.h>
2320010 #include <sys/types.h>          // gid_t
2320011
2320012 //-------------------------------------------------------------------------
2320013 struct group {
2320014     char   *gr_name;
2320015     gid_t   gr_gid;
2320016     char  **gr_mem;
2320017 };
2320018 //-------------------------------------------------------------------------
2320019 struct group *getgrgid (gid_t gid);
2320020 struct group *getgrnam (const char *name);
2320021 //-------------------------------------------------------------------------
2320022
2320023 #endif
```

lib/grp/getgrgid.c

«

Si veda la sezione u0.2.

```
2330001 #include <grp.h>
2330002 #include <NULL.h>
2330003 //-------------------------------------------------------------------------
2330004 struct group *
2330005 getgrgid (gid_t gid)
2330006 {
2330007     static char *name = "none";
2330008     static struct group grp;
2330009     //
2330010     // os16 does not have a group management, so the answare is always
2330011     // the same.
2330012     //
2330013     grp.gr_name = name;
2330014     grp.gr_gid  = (gid_t) -1;
2330015     grp.gr_mem  = NULL;
2330016     //
2330017     return (&grp);
2330018 }
```

lib/grp/getgrnam.c

«

Si veda la sezione u0.2.

```
2340001 #include <grp.h>
2340002 //-------------------------------------------------------------------------
2340003 struct group *
2340004 getgrnam (const char *name)
2340005 {
2340006     return (getgrgid ((gid_t) 0));
2340007 }
```

os16: «lib/libgen.h»

«

Si veda la sezione u0.2.

```
2350001 #ifndef _LIBGEN_H
2350002 #define _LIBGEN_H       1
2350003
2350004 //-------------------------------------------------------------------------
2350005 char *basename (char *path);
2350006 char *dirname  (char *path);
2350007 //-------------------------------------------------------------------------
2350008
2350009 #endif
```

lib/libgen/basename.c

«

Si veda la sezione u0.7.

```
2360001 #include <libgen.h>
2360002 #include <limits.h>
2360003 #include <stddef.h>
2360004 #include <string.h>
2360005 //-------------------------------------------------------------------------
2360006 char *
2360007 basename (char *path)
2360008 {
2360009     static char  *point = ".";          // When 'path' is NULL.
```

```
2360010             char  *p;                    // Pointer inside ‘path’.
2360011             int    i;                    // Scan index inside ‘path’.
2360012       //
2360013       // Empty path.
2360014       //
2360015       if (path == NULL || strlen (path) == 0)
2360016         {
2360017           return (point);
2360018         }
2360019       //
2360020       // Remove all final ‘/’ if it exists, excluded the first character:
2360021       // ‘i’ is kept greater than zero.
2360022       //
2360023       for (i = (strlen (path) - 1); i > 0 && path[i] == ’/’; i--)
2360024         {
2360025           path[i] = 0;
2360026         }
2360027       //
2360028       // After removal of extra final ‘/’, if there is only one ‘/’, this
2360029       // is to be returned.
2360030       //
2360031       if (strncmp (path, "/",  PATH_MAX) == 0)
2360032         {
2360033           return (path);
2360034         }
2360035       //
2360036       // If there are no ‘/’.
2360037       //
2360038       if (strchr (path, ’/’) == NULL)
2360039         {
2360040           return (path);
2360041         }
2360042       //
2360043       // Find the last ‘/’ and calculate a pointer to the base name.
2360044       //
2360045       p = strrchr (path, (unsigned int) ’/’);
2360046       p++;
2360047       //
2360048       // Return the pointer to the base name.
2360049       //
2360050       return (p);
2360051   }
```

lib/libgen/dirname.c

«

Si veda la sezione u0.7.

```
2370001   #include <libgen.h>
2370002   #include <limits.h>
2370003   #include <stddef.h>
2370004   #include <string.h>
2370005   //-----------------------------------------------------------------------
2370006   char *
2370007   dirname (char *path)
2370008   {
2370009       static char  *point = ".";          // When ‘path’ is NULL.
2370010              char  *p;                    // Pointer inside ‘path’.
2370011              int    i;                    // Scan index inside ‘path’.
2370012       //
2370013       // Empty path.
2370014       //
2370015       if      (path == NULL || strlen (path) == 0)
2370016         {
2370017           return (point);
2370018         }
2370019       //
2370020       // Simple cases.
2370021       //
2370022       if (strncmp (path, "/",  PATH_MAX) == 0  ||
2370023           strncmp (path, ".",  PATH_MAX) == 0  ||
2370024           strncmp (path, "..", PATH_MAX) == 0)
2370025         {
2370026           return (path);
2370027         }
2370028       //
2370029       // Remove all final ‘/’ if it exists, excluded the first character:
2370030       // ‘i’ is kept greater than zero.
2370031       //
2370032       for (i = (strlen (path) - 1); i > 0 && path[i] == ’/’; i--)
2370033         {
2370034           path[i] = 0;
2370035         }
2370036       //
2370037       // After removal of extra final ‘/’, if there is only one ‘/’, this
2370038       // is to be returned.
2370039       //
2370040       if (strncmp (path, "/",  PATH_MAX) == 0)
2370041         {
2370042           return (path);
2370043         }
2370044       //
2370045       // If there are no ‘/’
2370046       //
2370047       if (strchr (path, ’/’) == NULL)
2370048         {
2370049           return (point);
2370050         }
2370051       //
2370052       // If there is only a ‘/’ a the beginning.
2370053       //
2370054       if (path[0] == ’/’                                  &&
2370055           strchr (&path[1], (unsigned int) ’/’) == NULL)
2370056         {
2370057           path[1] = 0;
2370058           return (path);
2370059         }
2370060       //
2370061       // Replace the last ‘/’ with zero.
2370062       //
2370063        p = strrchr (path, (unsigned int) ’/’);
2370064       *p = 0;
2370065       //
2370066       // Now remove extra duplicated final ‘/’, except the very first
2370067       // character: ‘i’ is kept greater than zero.
2370068       //
2370069       for (i = (strlen (path) - 1); i > 0 && path[i] == ’/’; i--)
2370070         {
2370071           path[i] = 0;
2370072         }
2370073       //
2370074       // Now ‘path’ appears as a reduced string: the original path string
2370075       // is modified.
2370076       //
2370077       return (path);
2370078   }
```

os16: «lib/pwd.h»

«

Si veda la sezione u0.2.

```
2380001   #ifndef _PWD_H
2380002   #define _PWD_H        1
2380003
2380004   #include <const.h>
2380005   #include <restrict.h>
2380006   #include <sys/types.h>          // gid_t, uid_t
2380007   //-----------------------------------------------------------------------
2380008   struct passwd {
2380009       char  *pw_name;
2380010       char  *pw_passwd;
2380011       uid_t  pw_uid;
2380012       gid_t  pw_gid;
2380013       char  *pw_gecos;
2380014       char  *pw_dir;
2380015       char  *pw_shell;
2380016   };
2380017   //-----------------------------------------------------------------------
2380018   struct passwd *getpwent (void);
2380019   void           setpwent (void);
2380020   void           endpwent (void);
2380021   struct passwd *getpwnam (const char *name);
2380022   struct passwd *getpwuid (uid_t uid);
2380023   //-----------------------------------------------------------------------
2380024
2380025   #endif
```

lib/pwd/pwent.c

«

Si veda la sezione u0.53.

```
2390001   #include <pwd.h>
2390002   #include <stdio.h>
2390003   #include <string.h>
2390004   #include <stdlib.h>
2390005
2390006   //-----------------------------------------------------------------------
2390007   static char          buffer[BUFSIZ];
2390008   static struct passwd pw;
2390009   static FILE         *fp = NULL;
2390010   //-----------------------------------------------------------------------
2390011   struct passwd *
2390012   getpwent (void)
2390013   {
2390014       void *pstatus;
2390015       char *char_uid;
2390016       char *char_gid;
2390017       //
2390018       if (fp == NULL)
2390019         {
2390020           fp = fopen ("/etc/passwd", "r");
2390021           if (fp == NULL)
2390022             {
2390023               return NULL;
2390024             }
2390025         }
2390026       //
2390027       pstatus = fgets (buffer, BUFSIZ, fp);
2390028       if (pstatus == NULL)
2390029         {
2390030           return (NULL);
2390031         }
2390032       //
2390033       pw.pw_name   = strtok (buffer, ":");
2390034       pw.pw_passwd = strtok (NULL, ":");
2390035       char_uid     = strtok (NULL, ":");
2390036       char_gid     = strtok (NULL, ":");
2390037       pw.pw_gecos  = strtok (NULL, ":");
2390038       pw.pw_dir    = strtok (NULL, ":");
2390039       pw.pw_shell  = strtok (NULL, ":");
2390040       pw.pw_uid    = (uid_t) atoi (char_uid);
```

```
2390041     pw.pw_gid    = (gid_t) atoi (char_gid);
2390042     //
2390043     return (&pw);
2390044 }
2390045 //---------------------------------------------------------------------
2390046 void
2390047 endpwent (void)
2390048 {
2390049     int status;
2390050     //
2390051     if (fp != NULL)
2390052       {
2390053         fclose (fp);
2390054         if (status != NULL)
2390055           {
2390056             fp = NULL;
2390057           }
2390058       }
2390059 }
2390060 //---------------------------------------------------------------------
2390061 void
2390062 setpwent (void)
2390063 {
2390064     if (fp != NULL)
2390065       {
2390066         rewind (fp);
2390067       }
2390068 }
2390069 //---------------------------------------------------------------------
2390070 struct passwd *
2390071 getpwnam (const char *name)
2390072 {
2390073     struct passwd *pw;
2390074     //
2390075     setpwent ();
2390076     //
2390077     for (;;)
2390078       {
2390079         pw = getpwent ();
2390080         if (pw == NULL)
2390081           {
2390082             return (NULL);
2390083           }
2390084         if (strcmp (pw->pw_name, name) == 0)
2390085           {
2390086             return (pw);
2390087           }
2390088       }
2390089 }
2390090 //---------------------------------------------------------------------
2390091 struct passwd *
2390092 getpwuid (uid_t uid)
2390093 {
2390094     struct passwd *pw;
2390095     //
2390096     setpwent ();
2390097     //
2390098     for (;;)
2390099       {
2390100         pw = getpwent ();
2390101         if (pw == NULL)
2390102           {
2390103             return (NULL);
2390104           }
2390105         if (pw->pw_uid == uid)
2390106           {
2390107             return (pw);
2390108           }
2390109       }
2390110 }
```

os16: «lib/signal.h»

«

Si veda la sezione u0.2.

```
2400001 #ifndef _SIGNAL_H
2400002 #define _SIGNAL_H       1
2400003
2400004 #include <sys/types.h>
2400005 //---------------------------------------------------------------------
2400006 #define SIGHUP           1
2400007 #define SIGINT           2
2400008 #define SIGQUIT          3
2400009 #define SIGILL           4
2400010 #define SIGABRT          6
2400011 #define SIGFPE           8
2400012 #define SIGKILL          9
2400013 #define SIGSEGV         11
2400014 #define SIGPIPE         13
2400015 #define SIGALRM         14
2400016 #define SIGTERM         15
2400017 #define SIGSTOP         17
2400018 #define SIGTSTP         18
2400019 #define SIGCONT         19
2400020 #define SIGCHLD         20
2400021 #define SIGTTIN         21
2400022 #define SIGTTOU         22
2400023 #define SIGUSR1         30
2400024 #define SIGUSR2         31
2400025 //---------------------------------------------------------------------
2400026 typedef int sig_atomic_t;
```



```
2400027 typedef void (*sighandler_t) (int); // The type 'sighandler_t' is a
2400028                                     // pointer to a function for the
2400029                                     // signal handling, with a parameter
2400030                                     // of type 'int', returning 'void'.
2400031 //
2400032 // Special undeclarable functions.
2400033 //
2400034 #define SIG_ERR ((sighandler_t) -1) // It transform an integer number
2400035 #define SIG_DFL ((sighandler_t) 0)  // into a 'sighandler_t' type,
2400036 #define SIG_IGN ((sighandler_t) 1)  // that is, a pointer to a function
2400037                                     // that does not exists really.
2400038 //---------------------------------------------------------------------
2400039 sighandler_t signal (int sig, sighandler_t handler);
2400040 int          kill   (pid_t pid, int sig);
2400041 int          raise  (int sig);
2400042 //---------------------------------------------------------------------
2400043
2400044 #endif
```

lib/signal/kill.c

«

Si veda la sezione u0.22.

```
2410001 #include <sys/os16.h>
2410002 #include <sys/types.h>
2410003 #include <signal.h>
2410004 #include <errno.h>
2410005 #include <string.h>
2410006 //---------------------------------------------------------------------
2410007 int
2410008 kill (pid_t pid, int sig)
2410009 {
2410010     sysmsg_kill_t msg;
2410011     if (pid < -1)       // Currently unsupported.
2410012       {
2410013         errset (ESRCH);
2410014         return (-1);
2410015       }
2410016     msg.pid    = pid;
2410017     msg.signal = sig;
2410018     msg.ret    = 0;
2410019     msg.errno  = 0;
2410020     sys (SYS_KILL, &msg, (sizeof msg));
2410021     errno = msg.errno;
2410022     errln = msg.errln;
2410023     strncpy (errfn, msg.errfn, PATH_MAX);
2410024     return (msg.ret);
2410025 }
```

lib/signal/signal.c

«

Si veda la sezione u0.34.

```
2420001 #include <sys/os16.h>
2420002 #include <sys/types.h>
2420003 #include <signal.h>
2420004 #include <errno.h>
2420005 #include <string.h>
2420006 //---------------------------------------------------------------------
2420007 sighandler_t
2420008 signal (int sig, sighandler_t handler)
2420009 {
2420010     sysmsg_signal_t msg;
2420011
2420012     msg.signal  = sig;
2420013     msg.handler = handler;
2420014     msg.ret     = SIG_DFL;
2420015     msg.errno   = 0;
2420016     sys (SYS_SIGNAL, &msg, (sizeof msg));
2420017     errno = msg.errno;
2420018     errln = msg.errln;
2420019     strncpy (errfn, msg.errfn, PATH_MAX);
2420020     return (msg.ret);
2420021 }
```

os16: «lib/stdio.h»

«

Si veda la sezione u0.103.

```
2430001 #ifndef _STDIO_H
2430002 #define _STDIO_H        1
2430003
2430004 #include <const.h>
2430005 #include <restrict.h>
2430006 #include <stdarg.h>
2430007 #include <stdint.h>
2430008 #include <limits.h>
2430009 #include <NULL.h>
2430010 #include <size_t.h>
2430011 #include <sys/types.h>
2430012 #include <SEEK.h>       // SEEK_CUR, SEEK_SET, SEEK_END
2430013 //---------------------------------------------------------------------
2430014 #define BUFSIZ              2048  // Like the file system max zone
2430015                                   // size.
2430016 #define _IOFBF                 0  // Input-output fully buffered.
2430017 #define _IOLBF                 1  // Input-output line buffered.
2430018 #define _IONBF                 2  // Input-output with no buffering.
2430019
```

```
#define L_tmpnam    FILENAME_MAX  // <limits.h>

#define FOPEN_MAX       OPEN_MAX  // <limits.h>
#define FILENAME_MAX    NAME_MAX  // <limits.h>
#define TMP_MAX           0x7FFF

#define EOF                 (-1)  // Must be a negative value.
//---------------------------------------------------------------------
typedef off_t     fpos_t;   // 'off_t' defined in <sys/types.h>.

typedef struct {
    int           fdn;      // File descriptor number.
    char          error;    // Error indicator.
    char          eof;      // End of file indicator.
} FILE;

extern FILE _stream[];      // Defined inside 'lib/stdio/FILE.c'.

#define stdin   (&_stream[0])
#define stdout  (&_stream[1])
#define stderr  (&_stream[2])
//---------------------------------------------------------------------
void     clearerr   (FILE *fp);
int      fclose     (FILE *fp);
int      feof       (FILE *fp);
int      ferror     (FILE *fp);
int      fflush     (FILE *fp);
int      fgetc      (FILE *fp);
int      fgetpos    (FILE *restrict fp, fpos_t *restrict pos);
char    *fgets      (char *restrict string, int n, FILE *restrict fp);
int      fileno     (FILE *fp);
FILE    *fopen      (const char *path, const char *mode);
int      fprintf    (FILE *fp, char *restrict format, ...);
int      fputc      (int c, FILE *fp);
int      fputs      (const char *restrict string, FILE *restrict fp);
size_t   fread      (void *restrict buffer, size_t size, size_t nmemb,
                     FILE *restrict fp);
FILE    *freopen    (const char *restrict path,
                     const char *restrict mode,
                     FILE *restrict fp);
int      fscanf     (FILE *restrict fp, const char *restrict format,
                     ...);
int      fseek      (FILE *fp, long int offset, int whence);
int      fsetpos    (FILE *fp, const fpos_t *pos);
long int ftell      (FILE *fp);
off_t    ftello     (FILE *fp);
size_t   fwrite     (const void *restrict buffer, size_t size,
                     size_t nmemb, FILE *restrict fp);
#define  getc(p)    (fgetc (p))
int      getchar    (void);
char    *gets       (char *string);
void     perror     (const char *string);
int      printf     (const char *restrict format, ...);
#define  putc(c, p) (fputc ((c), (p)))
#define  putchar(c) (fputc ((c), (stdout)))
int      puts       (const char *string);
void     rewind     (FILE *fp);
int      scanf      (const char *restrict format, ...);
void     setbuf     (FILE *restrict fp, char *restrict buffer);
int      setvbuf    (FILE *restrict fp, char *restrict buffer,
                     int buf_mode, size_t size);
int      snprintf   (char *restrict string, size_t size,
                     const char *restrict format, ...);
int      sprintf    (char *restrict string, const char *restrict format,
                     ...);
int      sscanf     (char *restrict string, const char *restrict format,
                     ...);
int      vfprintf   (FILE *fp, char *restrict format, va_list arg);
int      vfscanf    (FILE *restrict fp, const char *restrict format,
                     va_list arg);
int      vprintf    (const char *restrict format, va_list arg);
int      vscanf     (const char *restrict format, va_list ap);
int      vsnprintf  (char *restrict string, size_t size,
                     const char *restrict format, va_list arg);
int      vsprintf   (char *restrict string, const char *restrict format,
                     va_list arg);
int      vsscanf    (const char *string, const char *format,
                     va_list ap);

#endif
```

lib/stdio/FILE.c

«

Si veda la sezione u0.2.

```
#include <stdio.h>
//
// There must be room for at least 'FOPEN_MAX' elements.
//
FILE _stream[FOPEN_MAX];
//---------------------------------------------------------------------
void
_stdio_stream_setup (void)
{
    _stream[0].fdn   = 0;
    _stream[0].error = 0;
    _stream[0].eof   = 0;

    _stream[1].fdn   = 1;
    _stream[1].error = 0;
    _stream[1].eof   = 0;

    _stream[2].fdn   = 2;
    _stream[2].error = 0;
    _stream[2].eof   = 0;
}
```

lib/stdio/clearerr.c

«

Si veda la sezione u0.9.

```
#include <stdio.h>
//---------------------------------------------------------------------
void
clearerr (FILE *fp)
{
    if (fp != NULL)
      {
        fp->error = 0;
        fp->eof   = 0;
      }
}
```

lib/stdio/fclose.c

«

Si veda la sezione u0.27.

```
#include <stdio.h>
#include <unistd.h>
//---------------------------------------------------------------------
int
fclose (FILE *fp)
{
    return (close (fp->fdn));
}
```

lib/stdio/feof.c

«

Si veda la sezione u0.28.

```
#include <stdio.h>
//---------------------------------------------------------------------
int
feof (FILE *fp)
{
    if (fp != NULL)
      {
        return (fp->eof);
      }
    return (0);
}
```

lib/stdio/ferror.c

«

Si veda la sezione u0.29.

```
#include <stdio.h>
//---------------------------------------------------------------------
int
ferror (FILE *fp)
{
    if (fp != NULL)
      {
        return (fp->error);
      }
    return (0);
}
```

lib/stdio/fflush.c

«

Si veda la sezione u0.30.

```
#include <stdio.h>
//---------------------------------------------------------------------
int
fflush (FILE *fp)
{
    //
    // The os16 library does not have any buffered data.
    //
    return (0);
}
```

lib/stdio/fgetc.c

«

Si veda la sezione u0.31.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
//---------------------------------------------------------------------
int
fgetc (FILE *fp)
{
    ssize_t size_read;
    int     c;          // Character read.
```

```
2500010     //
2500011     for (c = 0;;)
2500012       {
2500013         size_read = read (fp->fdn, &c, (size_t) 1);
2500014         //
2500015         if (size_read <= 0)
2500016           {
2500017             //
2500018             // It is the end of file (zero) otherwise there is a
2500019             // problem (a negative value): return 'EOF'.
2500020             //
2500021             return (EOF);
2500022           }
2500023         //
2500024         // Valid read: end of scan.
2500025         //
2500026         return (c);
2500027       }
2500028 }
```

lib/stdio/fgetpos.c

« Si veda la sezione u0.32.

```
2510001 #include <stdio.h>
2510002 //---------------------------------------------------------------------
2510003 int
2510004 fgetpos (FILE *restrict fp, fpos_t *restrict pos)
2510005 {
2510006     long int position;
2510007     //
2510008     if (fp != NULL)
2510009       {
2510010         position = ftell (fp);
2510011         if (position >= 0)
2510012           {
2510013             *pos = position;
2510014             return (0);
2510015           }
2510016       }
2510017     return (-1);
2510018 }
```

lib/stdio/fgets.c

« Si veda la sezione u0.33.

```
2520001 #include <stdio.h>
2520002 #include <sys/types.h>
2520003 #include <unistd.h>
2520004 #include <stddef.h>
2520005 //---------------------------------------------------------------------
2520006 char *
2520007 fgets (char *restrict string, int n, FILE *restrict fp)
2520008 {
2520009     ssize_t size_read;
2520010     int     b;          // Index inside the string buffer.
2520011     //
2520012     for (b = 0; b < (n-1); b++, string[b] = 0)
2520013       {
2520014         size_read = read (fp->fdn, &string[b], (size_t) 1);
2520015         //
2520016         if (size_read <= 0)
2520017           {
2520018             //
2520019             // It is the end of file (zero) otherwise there is a
2520020             // problem (a negative value).
2520021             //
2520022             string[b] = 0;
2520023             break;
2520024           }
2520025         //
2520026         if (string[b] == '\n')
2520027           {
2520028             b++;
2520029             string[b] = 0;
2520030             break;
2520031           }
2520032       }
2520033     //
2520034     // If 'b' is zero, nothing was read and 'NULL' is returned.
2520035     //
2520036     if (b == 0)
2520037       {
2520038         return (NULL);
2520039       }
2520040     else
2520041       {
2520042         return (string);
2520043       }
2520044 }
```

lib/stdio/fileno.c

« Si veda la sezione u0.34.

```
2530001 #include <stdio.h>
2530002 #include <errno.h>
```



```
2530003 //---------------------------------------------------------------------
2530004 int
2530005 fileno (FILE *fp)
2530006 {
2530007     if (fp != NULL)
2530008       {
2530009         return (fp->fdn);
2530010       }
2530011     errset (EBADF);                     // Bad file descriptor.
2530012     return (-1);
2530013 }
```

lib/stdio/fopen.c

« Si veda la sezione u0.35.

```
2540001 #include <fcntl.h>
2540002 #include <stdarg.h>
2540003 #include <stddef.h>
2540004 #include <string.h>
2540005 #include <errno.h>
2540006 #include <sys/os16.h>
2540007 #include <const.h>
2540008 #include <limits.h>
2540009 #include <stdio.h>
2540010
2540011 //---------------------------------------------------------------------
2540012 FILE *
2540013 fopen (const char *path, const char *mode)
2540014 {
2540015     int fdn;
2540016     //
2540017     if      (strcmp (mode, "r")   ||
2540018              strcmp (mode, "rb"))
2540019       {
2540020         fdn = open (path, O_RDONLY);
2540021       }
2540022     else if (strcmp (mode, "r+")  ||
2540023              strcmp (mode, "r+b") ||
2540024              strcmp (mode, "rb+"))
2540025       {
2540026         fdn = open (path, O_RDWR);
2540027       }
2540028     else if (strcmp (mode, "w")  ||
2540029              strcmp (mode, "wb"))
2540030       {
2540031         fdn = open (path, O_WRONLY|O_CREAT|O_TRUNC, 0666);
2540032       }
2540033     else if (strcmp (mode, "w+")  ||
2540034              strcmp (mode, "w+b") ||
2540035              strcmp (mode, "wb+"))
2540036       {
2540037         fdn = open (path, O_RDWR|O_CREAT|O_TRUNC, 0666);
2540038       }
2540039     else if (strcmp (mode, "a")  ||
2540040              strcmp (mode, "ab"))
2540041       {
2540042         fdn = open (path, O_WRONLY|O_APPEND|O_CREAT|O_TRUNC, 0666);
2540043       }
2540044     else if (strcmp (mode, "a+")  ||
2540045              strcmp (mode, "a+b") ||
2540046              strcmp (mode, "ab+"))
2540047       {
2540048         fdn = open (path, O_RDWR|O_APPEND|O_CREAT|O_TRUNC, 0666);
2540049       }
2540050     else
2540051       {
2540052         errset (EINVAL);                // Invalid argument.
2540053         return (NULL);
2540054       }
2540055     //
2540056     // Check the file descriptor returned.
2540057     //
2540058     if (fdn < 0)
2540059       {
2540060         //
2540061         // The variable 'errno' is already set.
2540062         //
2540063         errset (errno);
2540064         return (NULL);
2540065       }
2540066     //
2540067     // A valid file descriptor is available: convert it into a file
2540068     // stream. Please note that the file descriptor number must be
2540069     // saved inside the corresponding '_stream[]' array, because the
2540070     // file pointer do not have knowledge of the relative position
2540071     // inside the array.
2540072     //
2540073     _stream[fdn].fdn = fdn;     // Saved the file descriptor number.
2540074     //
2540075     return (&_stream[fdn]);     // Returned the file stream pointer.
2540076 }
```

lib/stdio/fprintf.c

« Si veda la sezione u0.78.

```
2550001 #include <stdio.h>
2550002
2550003 //---------------------------------------------------------------------
```

```
2550004 int
2550005 fprintf (FILE *fp, char *restrict format, ...)
2550006 {
2550007     va_list ap;
2550008     va_start (ap, format);
2550009     return (vfprintf (fp, format, ap));
2550010 }
```

lib/stdio/fputc.c

«

Si veda la sezione u0.37.

```
2560001 #include <stdio.h>
2560002 #include <sys/types.h>
2560003 #include <sys/os16.h>
2560004 #include <string.h>
2560005 #include <unistd.h>
2560006 //---------------------------------------------------------------------
2560007 int
2560008 fputc (int c, FILE *fp)
2560009 {
2560010     ssize_t size_written;
2560011     char    character = (char) c;
2560012     size_written = write (fp->fdn, &character, (size_t) 1);
2560013     if (size_written < 0)
2560014       {
2560015         fp->eof = 1;
2560016         return (EOF);
2560017       }
2560018     return (c);
2560019 }
```

lib/stdio/fputs.c

«

Si veda la sezione u0.38.

```
2570001 #include <stdio.h>
2570002 #include <string.h>
2570003 //---------------------------------------------------------------------
2570004 int
2570005 fputs (const char *restrict string, FILE *restrict fp)
2570006 {
2570007     int      i; // Index inside the string to be printed.
2570008     int status;
2570009
2570010     for (i = 0; i < strlen (string); i++)
2570011       {
2570012         status = fputc (string[i], fp);
2570013         if (status == EOF)
2570014           {
2570015             fp->eof = 1;
2570016             return (EOF);
2570017           }
2570018       }
2570019     return (0);
2570020 }
```

lib/stdio/fread.c

«

Si veda la sezione u0.39.

```
2580001 #include <unistd.h>
2580002 #include <stdio.h>
2580003 //---------------------------------------------------------------------
2580004 size_t
2580005 fread (void *restrict buffer, size_t size, size_t nmemb,
2580006        FILE *restrict fp)
2580007 {
2580008     ssize_t size_read;
2580009     size_read = read (fp->fdn, buffer, (size_t) (size * nmemb));
2580010     if (size_read == 0)
2580011       {
2580012         fp->eof = 1;
2580013         return ((size_t) 0);
2580014       }
2580015     else if (size_read < 0)
2580016       {
2580017         fp->error = 1;
2580018         return ((size_t) 0);
2580019       }
2580020     else
2580021       {
2580022         return ((size_t) (size_read / size));
2580023       }
2580024 }
```

lib/stdio/freopen.c

«

Si veda la sezione u0.35.

```
2590001 #include <fcntl.h>
2590002 #include <stdarg.h>
2590003 #include <stddef.h>
2590004 #include <string.h>
2590005 #include <errno.h>
2590006 #include <sys/os16.h>
2590007 #include <const.h>
```



```
2590008 #include <limits.h>
2590009 #include <stdio.h>
2590010
2590011 //---------------------------------------------------------------------
2590012 FILE *
2590013 freopen (const char *restrict path, const char *restrict mode,
2590014          FILE *restrict fp)
2590015 {
2590016     int   status;
2590017     FILE *fp_new;
2590018     //
2590019     if (fp == NULL)
2590020       {
2590021         return (NULL);
2590022       }
2590023     //
2590024     status = fclose (fp);
2590025     if (status != 0)
2590026       {
2590027         fp->error = 1;
2590028         return (NULL);
2590029       }
2590030     //
2590031     fp_new = fopen (path, mode);
2590032     //
2590033     if (fp_new == NULL)
2590034       {
2590035         return (NULL);
2590036       }
2590037     //
2590038     if (fp_new != fp)
2590039       {
2590040         fclose (fp_new);
2590041         return (NULL);
2590042       }
2590043     //
2590044     return (fp_new);
2590045 }
```

lib/stdio/fscanf.c

«

Si veda la sezione u0.90.

```
2600001 #include <stdio.h>
2600002 //---------------------------------------------------------------------
2600003 int
2600004 fscanf (FILE *restrict fp, const char *restrict format, ...)
2600005 {
2600006     va_list ap;
2600007     va_start (ap, format);
2600008     return vfscanf (fp, format, ap);
2600009 }
```

lib/stdio/fseek.c

«

Si veda la sezione u0.43.

```
2610001 #include <stdio.h>
2610002 #include <unistd.h>
2610003 //---------------------------------------------------------------------
2610004 int
2610005 fseek (FILE *fp, long int offset, int whence)
2610006 {
2610007     off_t off_new;
2610008     off_new = lseek (fp->fdn, (off_t) offset, whence);
2610009     if (off_new < 0)
2610010       {
2610011         fp->error = 1;
2610012         return (-1);
2610013       }
2610014     else
2610015       {
2610016         fp->eof = 0;
2610017         return (0);
2610018       }
2610019 }
```

lib/stdio/fseeko.c

«

Si veda la sezione u0.43.

```
2620001 #include <stdio.h>
2620002 #include <unistd.h>
2620003 //---------------------------------------------------------------------
2620004 int
2620005 fseeko (FILE *fp, off_t offset, int whence)
2620006 {
2620007     off_t off_new;
2620008     off_new = lseek (fp->fdn, offset, whence);
2620009     if (off_new < 0)
2620010       {
2620011         fp->error = 1;
2620012         return (-1);
2620013       }
2620014     else
2620015       {
2620016         return (0);
2620017       }
2620018 }
```

lib/stdio/fsetpos.c

«

Si veda la sezione u0.32.

```
2630001  #include <stdio.h>
2630002  //------------------------------------------------------------------------
2630003  int
2630004  fsetpos (FILE *restrict fp, fpos_t *restrict pos)
2630005  {
2630006      long int position;
2630007      //
2630008      if (fp != NULL)
2630009        {
2630010          position = fseek (fp, (long int) *pos, SEEK_SET);
2630011          if (position >= 0)
2630012            {
2630013              *pos = position;
2630014              return (0);
2630015            }
2630016        }
2630017      return (-1);
2630018  }
```

lib/stdio/ftell.c

«

Si veda la sezione u0.46.

```
2640001  #include <stdio.h>
2640002  #include <unistd.h>
2640003  //------------------------------------------------------------------------
2640004  long int
2640005  ftell (FILE *fp)
2640006  {
2640007      return ((long int) lseek (fp->fdn, (off_t) 0, SEEK_CUR));
2640008  }
```

lib/stdio/ftello.c

«

Si veda la sezione u0.46.

```
2650001  #include <stdio.h>
2650002  #include <unistd.h>
2650003  //------------------------------------------------------------------------
2650004  off_t
2650005  ftello (FILE *fp)
2650006  {
2650007      return (lseek (fp->fdn, (off_t) 0, SEEK_CUR));
2650008  }
```

lib/stdio/fwrite.c

«

Si veda la sezione u0.48.

```
2660001  #include <unistd.h>
2660002  #include <stdio.h>
2660003  //------------------------------------------------------------------------
2660004  size_t
2660005  fwrite (const void *restrict buffer, size_t size, size_t nmemb,
2660006          FILE *restrict fp)
2660007  {
2660008      ssize_t size_written;
2660009      size_written = write (fp->fdn, buffer, (size_t) (size * nmemb));
2660010      if (size_written < 0)
2660011        {
2660012          fp->error = 1;
2660013          return ((size_t) 0);
2660014        }
2660015      else
2660016        {
2660017          return ((size_t) (size_written / size));
2660018        }
2660019  }
```

lib/stdio/getchar.c

«

Si veda la sezione u0.31.

```
2670001  #include <stdio.h>
2670002  #include <sys/types.h>
2670003  #include <unistd.h>
2670004  //------------------------------------------------------------------------
2670005  int
2670006  getchar (void)
2670007  {
2670008      ssize_t size_read;
2670009      int     c;          // Character read.
2670010      //
2670011      for (c = 0;;)
2670012        {
2670013          size_read = read (STDIN_FILENO, &c, (size_t) 1);
2670014          //
2670015          if (size_read <= 0)
2670016            {
2670017              //
2670018              // It is the end of file (zero) otherwise there is a
2670019              // problem (a negative value): return 'EOF'.
2670020              //
2670021              _stream[STDIN_FILENO].eof = 1;
2670022              return (EOF);
2670023            }
2670024          //
2670025          // Valid read.
2670026          //
2670027          if (size_read == 0)
2670028            {
2670029              //
2670030              // If no character is ready inside the keyboard buffer, just
2670031              // retry.
2670032              //
2670033              continue;
2670034            }
2670035          //
2670036          // End of scan.
2670037          //
2670038          return (c);
2670039        }
2670040  }
```


lib/stdio/gets.c

«

Si veda la sezione u0.33.

```
2680001  #include <stdio.h>
2680002  #include <sys/types.h>
2680003  #include <unistd.h>
2680004  #include <stddef.h>
2680005  //------------------------------------------------------------------------
2680006  char *
2680007  gets (char *string)
2680008  {
2680009      ssize_t size_read;
2680010      int     b;          // Index inside the string buffer.
2680011      //
2680012      for (b = 0;; b++, string[b] = 0)
2680013        {
2680014          size_read = read (STDIN_FILENO, &string[b], (size_t) 1);
2680015          //
2680016          if (size_read <= 0)
2680017            {
2680018              //
2680019              // It is the end of file (zero) otherwise there is a
2680020              // problem (a negative value).
2680021              //
2680022              _stream[STDIN_FILENO].eof = 1;
2680023              string[b] = 0;
2680024              break;
2680025            }
2680026          //
2680027          if (string[b] == '\n')
2680028            {
2680029              b++;
2680030              string[b] = 0;
2680031              break;
2680032            }
2680033        }
2680034      //
2680035      // If 'b' is zero, nothing was read and 'NULL' is returned.
2680036      //
2680037      if (b == 0)
2680038        {
2680039          return (NULL);
2680040        }
2680041      else
2680042        {
2680043          return (string);
2680044        }
2680045  }
```

lib/stdio/perror.c

«

Si veda la sezione u0.77.

```
2690001  #include <stdio.h>
2690002  #include <errno.h>
2690003  #include <stddef.h>
2690004  #include <string.h>
2690005  //------------------------------------------------------------------------
2690006  void
2690007  perror (const char *string)
2690008  {
2690009      //
2690010      // If errno is zero, there is nothing to show.
2690011      //
2690012      if (errno == 0)
2690013        {
2690014          return;
2690015        }
2690016      //
2690017      // Show the string if there is one.
2690018      //
2690019      if (string != NULL && strlen (string) > 0)
2690020        {
2690021          printf ("%s: ", string);
2690022        }
2690023      //
2690024      // Show the translated error.
2690025      //
2690026      if (errfn[0] != 0 && errln != 0)
```

```
2690027         {
2690028           printf ("[%s:%u:%i] %s\n",
2690029                   errfn, errln, errno, strerror (errno));
2690030         }
2690031       else
2690032         {
2690033           printf ("[%i] %s\n", errno, strerror (errno));
2690034         }
2690035 }
```

lib/stdio/printf.c

«

Si veda la sezione u0.78.

```
2700001 #include <stdio.h>
2700002 //-----------------------------------------------------------------------
2700003 int
2700004 printf (char *restrict format, ...)
2700005 {
2700006     va_list ap;
2700007     va_start (ap, format);
2700008     return (vprintf (format, ap));
2700009 }
```

lib/stdio/puts.c

«

Si veda la sezione u0.38.

```
2710001 #include <stdio.h>
2710002 //-----------------------------------------------------------------------
2710003 int
2710004 puts (const char *string)
2710005 {
2710006     int status;
2710007     status = printf ("%s\n", string);
2710008     if (status < 0)
2710009       {
2710010         return (EOF);
2710011       }
2710012     else
2710013       {
2710014         return (status);
2710015       }
2710016 }
```

lib/stdio/rewind.c

«

Si veda la sezione u0.88.

```
2720001 #include <stdio.h>
2720002 //-----------------------------------------------------------------------
2720003 void
2720004 rewind (FILE *fp)
2720005 {
2720006     (void) fseek (fp, 0L, SEEK_SET);
2720007     fp->error = 0;
2720008 }
```

lib/stdio/scanf.c

«

Si veda la sezione u0.90.

```
2730001 #include <stdio.h>
2730002 //-----------------------------------------------------------------------
2730003 int
2730004 scanf (const char *restrict format, ...)
2730005 {
2730006     va_list ap;
2730007     va_start (ap, format);
2730008     return vfscanf (stdin, format, ap);
2730009 }
```

lib/stdio/setbuf.c

«

Si veda la sezione u0.93.

```
2740001 #include <stdio.h>
2740002 //-----------------------------------------------------------------------
2740003 void
2740004 setbuf (FILE *restrict fp, char *restrict buffer)
2740005 {
2740006     //
2740007     // The os16 library does not have any buffered data.
2740008     //
2740009     return;
2740010 }
```

lib/stdio/setvbuf.c

«

Si veda la sezione u0.93.

```
2750001 #include <stdio.h>
2750002 //-----------------------------------------------------------------------
2750003 int
2750004 setvbuf (FILE *restrict fp, char *restrict buffer, int buf_mode,
2750005           size_t size)
2750006 {
2750007     //
2750008     // The os16 library does not have any buffered data.
2750009     //
2750010     return (0);
2750011 }
```


lib/stdio/snprintf.c

«

Si veda la sezione u0.78.

```
2760001 #include <stdio.h>
2760002 #include <stdarg.h>
2760003 //-----------------------------------------------------------------------
2760004 int
2760005 snprintf (char *restrict string, size_t size,
2760006           const char *restrict format, ...)
2760007 {
2760008     va_list ap;
2760009     va_start (ap, format);
2760010     return vsnprintf (string, size, format, ap);
2760011 }
```

lib/stdio/sprintf.c

«

Si veda la sezione u0.78.

```
2770001 #include <stdio.h>
2770002 #include <stdarg.h>
2770003 //-----------------------------------------------------------------------
2770004 int
2770005 sprintf (char *restrict string, const char *restrict format,
2770006           ...)
2770007 {
2770008     va_list ap;
2770009     va_start (ap, format);
2770010     return vsnprintf (string, (size_t) BUFSIZ, format, ap);
2770011 }
```

lib/stdio/sscanf.c

«

Si veda la sezione u0.90.

```
2780001 #include <stdio.h>
2780002 //-----------------------------------------------------------------------
2780003 int
2780004 sscanf (char *restrict string, const char *restrict format, ...)
2780005 {
2780006     va_list ap;
2780007     va_start (ap, format);
2780008     return vsscanf (string, format, ap);
2780009 }
```

lib/stdio/vfprintf.c

«

Si veda la sezione u0.128.

```
2790001 #include <stdio.h>
2790002 #include <sys/types.h>
2790003 #include <sys/os16.h>
2790004 #include <string.h>
2790005 #include <unistd.h>
2790006 //-----------------------------------------------------------------------
2790007 int
2790008 vfprintf (FILE *fp, char *restrict format, va_list arg)
2790009 {
2790010     ssize_t     size_written;
2790011     size_t      size;
2790012     size_t      size_total;
2790013     int         status;
2790014     char        string[BUFSIZ];
2790015     char       *buffer = string;
2790016     //
2790017     buffer[0] = 0;
2790018     status    = vsprintf (buffer, format, arg);
2790019     //
2790020     size = strlen (buffer);
2790021     if (size >= BUFSIZ)
2790022       {
2790023         size = BUFSIZ;
2790024       }
2790025     //
2790026     for (size_total = 0, size_written = 0;
2790027          size_total < size;
2790028          size_total += size_written, buffer += size_written)
2790029       {
2790030         size_written = write (fp->fdn, buffer, size - size_total);
2790031         if (size_written < 0)
2790032           {
2790033             return (size_total);
2790034           }
2790035       }
2790036     return (size);
2790037 }
```

lib/stdio/vfscanf.c

«

Si veda la sezione u0.129.

```
2800001 #include <stdio.h>
2800002
2800003 //-----------------------------------------------------------------------
2800004 int vfsscanf (FILE *restrict fp, const char *string,
2800005              const char *restrict format, va_list ap);
2800006 //-----------------------------------------------------------------------
2800007 int
2800008 vfscanf (FILE *restrict fp, const char *restrict format, va_list ap)
2800009 {
2800010     return (vfsscanf (fp, NULL, format, ap));
2800011 }
2800012 //-----------------------------------------------------------------------
```

lib/stdio/vfsscanf.c

«

Si veda la sezione u0.129.

```
2810001 #include <stdint.h>
2810002 #include <stdbool.h>
2810003 #include <stdlib.h>
2810004 #include <string.h>
2810005 #include <stdio.h>
2810006 #include <stdarg.h>
2810007 #include <ctype.h>
2810008 #include <errno.h>
2810009 #include <stddef.h>
2810010 //-----------------------------------------------------------------------
2810011 //
2810012 // This function is not standard and is able to do the work of both
2810013 // 'vfscanf()' and 'vsscanf()'.
2810014 //
2810015 //-----------------------------------------------------------------------
2810016 #define WIDTH_MAX       64
2810017 //-----------------------------------------------------------------------
2810018 static intmax_t strtointmax (const char *restrict string,
2810019                              char **restrict endptr, int base,
2810020                              size_t max_width);
2810021 static int      ass_or_eof  (int consumed, int assigned);
2810022 //-----------------------------------------------------------------------
2810023 int
2810024 vfsscanf (FILE *restrict fp, const char *string,
2810025           const char *restrict format, va_list ap)
2810026 {
2810027     int                 f               = 0;            // Format index.
2810028     char                buffer[BUFSIZ];
2810029     const char          *input          = string;       // Default.
2810030     const char          *start          = input;        // Default.
2810031     char                *next           = NULL;
2810032     int                 scanned         = 0;
2810033     //
2810034     bool                stream          = 0;
2810035     bool                flag_star       = 0;
2810036     bool                specifier       = 0;
2810037     bool                specifier_flags = 0;
2810038     bool                specifier_width = 0;
2810039     bool                specifier_type  = 0;
2810040     bool                inverted        = 0;
2810041     //
2810042     char                *ptr_char;
2810043     signed char         *ptr_schar;
2810044     unsigned char       *ptr_uchar;
2810045     short int           *ptr_sshort;
2810046     unsigned short int  *ptr_ushort;
2810047     int                 *ptr_sint;
2810048     unsigned int        *ptr_uint;
2810049     long int            *ptr_slong;
2810050     unsigned long int   *ptr_ulong;
2810051     intmax_t            *ptr_simax;
2810052     uintmax_t           *ptr_uimax;
2810053     size_t              *ptr_size;
2810054     ptrdiff_t           *ptr_ptrdiff;
2810055     void                **ptr_void;
2810056     //
2810057     size_t              width;
2810058     char                width_string[WIDTH_MAX+1];
2810059     int                 w;              // Index inside width string.
2810060     int                 assigned        = 0;    // Assignment counter.
2810061     int                 consumed        = 0;    // Consumed counter.
2810062     //
2810063     intmax_t            value_i;
2810064     uintmax_t           value_u;
2810065     //
2810066     const char          *end_format;
2810067     const char          *end_input;
2810068     int                 count;          // Generic counter.
2810069     int                 index;          // Generic index.
2810070     bool                ascii[128];
2810071     //
2810072     void                *pstatus;
2810073     //
2810074     // Initialize some data.
2810075     //
2810076     width_string[0] = '\0';
2810077     end_format      = format + (strlen (format));
2810078     //
2810079     // Check arguments and find where input comes.
2810080     //
```



```
2810081     if (fp == NULL && (string == NULL || string[0] == 0))
2810082       {
2810083         errset (EINVAL);                // Invalid argument.
2810084         return (EOF);
2810085       }
2810086     //
2810087     if (fp != NULL && string != NULL && string[0] != 0)
2810088       {
2810089         errset (EINVAL);                // Invalid argument.
2810090         return (EOF);
2810091       }
2810092     //
2810093     if (fp != NULL)
2810094       {
2810095         stream = 1;
2810096       }
2810097     //
2810098     //
2810099     //
2810100     for (;;)
2810101       {
2810102         if (stream)
2810103           {
2810104             pstatus = fgets (buffer, BUFSIZ, fp);
2810105             //
2810106             if (pstatus == NULL)
2810107               {
2810108                 return (ass_or_eof (consumed, assigned));
2810109               }
2810110             //
2810111             input = buffer;
2810112             start = input;
2810113             next  = NULL;
2810114           }
2810115         //
2810116         // Calculate end input.
2810117         //
2810118         end_input = input + (strlen (input));
2810119         //
2810120         // Scan format and input strings. Index 'f' is not reset.
2810121         //
2810122         while (&format[f] < end_format && input < end_input)
2810123           {
2810124             if (!specifier)
2810125               {
2810126                 //--------------- The context is not inside a specifier.
2810127                 if (isspace (format[f]))
2810128                   {
2810129                     //------------------------------------------ Space.
2810130                     while (isspace (*input))
2810131                       {
2810132                         input++;
2810133                       }
2810134                     //
2810135                     // Verify that the input string is not finished.
2810136                     //
2810137                     if (input[0] == 0)
2810138                       {
2810139                         //
2810140                         // As the input string is finished, the format
2810141                         // string index is not advanced, because there
2810142                         // might be more spaces on the next line (if
2810143                         // there is a next line, of course).
2810144                         //
2810145                         continue;
2810146                       }
2810147                     else
2810148                       {
2810149                         f++;
2810150                         continue;
2810151                       }
2810152                   }
2810153                 if (format[f] != '%')
2810154                   {
2810155                     //---------------------------- Ordinary character.
2810156                     if (format[f] == *input)
2810157                       {
2810158                         input++;
2810159                         f++;
2810160                         continue;
2810161                       }
2810162                     else
2810163                       {
2810164                         return (ass_or_eof (consumed, assigned));
2810165                       }
2810166                   }
2810167                 if (format[f] == '%' && format[f+1] == '%')
2810168                   {
2810169                     //------------------------- Matching a literal '%'.
2810170                     f++;
2810171                     if (format[f] == *input)
2810172                       {
2810173                         input++;
2810174                         f++;
2810175                         continue;
2810176                       }
2810177                     else
2810178                       {
2810179                         return (ass_or_eof (consumed, assigned));
2810180                       }
2810181                   }
```

```
2810182                 if (format[f] == '%')
2810183                   {
2810184                     //-------------------------- Percent of a specifier.
2810185                     f++;
2810186                     specifier       = 1;
2810187                     specifier_flags = 1;
2810188                     continue;
2810189                   }
2810190                 }
2810191               //
2810192               if (specifier && specifier_flags)
2810193                 {
2810194                   //--------------- The context is inside specifier flags.
2810195                   if (format[f] == '*')
2810196                     {
2810197                       //---------------------- Assignment suppression star.
2810198                       flag_star = 1;
2810199                       f++;
2810200                     }
2810201                   else
2810202                     {
2810203                       //----- End of flags and begin of specifier length.
2810204                       specifier_flags = 0;
2810205                       specifier_width = 1;
2810206                     }
2810207                 }
2810208               //
2810209               if (specifier && specifier_width)
2810210                 {
2810211                   //------------- The context is inside a specifier width.
2810212                   for (w = 0;
2810213                        format[f] >= '0'
2810214                     && format[f] <= '9'
2810215                     && w < WIDTH_MAX;
2810216                        w++)
2810217                     {
2810218                       width_string[w] = format[f];
2810219                       f++;
2810220                     }
2810221                   width_string[w] = '\0';
2810222                   width = atoi (width_string);
2810223                   if (width > WIDTH_MAX)
2810224                     {
2810225                       width = WIDTH_MAX;
2810226                     }
2810227                   //
2810228                   // A zero width means an unspecified limit for the field
2810229                   // length.
2810230                   //
2810231                   //--------- End of spec. width and begin of spec. type.
2810232                   specifier_width = 0;
2810233                   specifier_type  = 1;
2810234                 }
2810235               //
2810236               if (specifier && specifier_type)
2810237                 {
2810238                   //
2810239                   // Specifiers with length modifier.
2810240                   //
2810241                   if (format[f] == 'h' && format[f+1] == 'h')
2810242                     {
2810243                       //---------------------------------------- char.
2810244                       if      (format[f+2] == 'd')
2810245                         {
2810246                           //----------------------- signed char, base 10.
2810247                           value_i = strtointmax (input, &next, 10, width);
2810248                           if (input == next)
2810249                             {
2810250                               return (ass_or_eof (consumed, assigned));
2810251                             }
2810252                           consumed++;
2810253                           if (!flag_star)
2810254                             {
2810255                               ptr_schar = va_arg (ap, signed char *);
2810256                              *ptr_schar = value_i;
2810257                               assigned++;
2810258                             }
2810259                           f += 3;
2810260                           input = next;
2810261                         }
2810262                       else if (format[f+2] == 'i')
2810263                         {
2810264                           //------------------- signed char, base unknown.
2810265                           value_i = strtointmax (input, &next, 0, width);
2810266                           if (input == next)
2810267                             {
2810268                               return (ass_or_eof (consumed, assigned));
2810269                             }
2810270                           consumed++;
2810271                           if (!flag_star)
2810272                             {
2810273                               ptr_schar = va_arg (ap, signed char *);
2810274                              *ptr_schar = value_i;
2810275                               assigned++;
2810276                             }
2810277                           f += 3;
2810278                           input = next;
2810279                         }
2810280                       else if (format[f+2] == 'o')
2810281                         {
2810282                           //------------------------- signed char, base 8.
```



```
2810283                           value_i = strtointmax (input, &next, 8, width);
2810284                           if (input == next)
2810285                             {
2810286                               return (ass_or_eof (consumed, assigned));
2810287                             }
2810288                           consumed++;
2810289                           if (!flag_star)
2810290                             {
2810291                               ptr_schar = va_arg (ap, signed char *);
2810292                              *ptr_schar = value_i;
2810293                               assigned++;
2810294                             }
2810295                           f += 3;
2810296                           input = next;
2810297                         }
2810298                       else if (format[f+2] == 'u')
2810299                         {
2810300                           //---------------------- unsigned char, base 10.
2810301                           value_u = strtointmax (input, &next, 10, width);
2810302                           if (input == next)
2810303                             {
2810304                               return (ass_or_eof (consumed, assigned));
2810305                             }
2810306                           consumed++;
2810307                           if (!flag_star)
2810308                             {
2810309                               ptr_uchar = va_arg (ap, unsigned char *);
2810310                              *ptr_uchar = value_u;
2810311                               assigned++;
2810312                             }
2810313                           f += 3;
2810314                           input = next;
2810315                         }
2810316                       else if (format[f+2] == 'x' || format[f+2] == 'X')
2810317                         {
2810318                           //------------------------ signed char, base 16.
2810319                           value_i = strtointmax (input, &next, 16, width);
2810320                           if (input == next)
2810321                             {
2810322                               return (ass_or_eof (consumed, assigned));
2810323                             }
2810324                           consumed++;
2810325                           if (!flag_star)
2810326                             {
2810327                               ptr_schar = va_arg (ap, signed char *);
2810328                              *ptr_schar = value_i;
2810329                               assigned++;
2810330                             }
2810331                           f += 3;
2810332                           input = next;
2810333                         }
2810334                       else if (format[f+2] == 'n')
2810335                         {
2810336                           //----------- signed char, string index counter.
2810337                           ptr_schar = va_arg (ap, signed char *);
2810338                          *ptr_schar = (signed char)
2810339                                       (input - start + scanned);
2810340                           f += 3;
2810341                         }
2810342                       else
2810343                         {
2810344                           //------------ unsupported or unknown specifier.
2810345                           f += 2;
2810346                         }
2810347                     }
2810348                   else if (format[f] == 'h')
2810349                     {
2810350                       //----------------------------------------- short.
2810351                       if      (format[f+1] == 'd')
2810352                         {
2810353                           //----------------------- signed short, base 10.
2810354                           value_i = strtointmax (input, &next, 10, width);
2810355                           if (input == next)
2810356                             {
2810357                               return (ass_or_eof (consumed, assigned));
2810358                             }
2810359                           consumed++;
2810360                           if (!flag_star)
2810361                             {
2810362                               ptr_sshort = va_arg (ap, signed short *);
2810363                              *ptr_sshort = value_i;
2810364                               assigned++;
2810365                             }
2810366                           f += 2;
2810367                           input = next;
2810368                         }
2810369                       else if (format[f+1] == 'i')
2810370                         {
2810371                           //------------------ signed short, base unknown.
2810372                           value_i = strtointmax (input, &next, 0, width);
2810373                           if (input == next)
2810374                             {
2810375                               return (ass_or_eof (consumed, assigned));
2810376                             }
2810377                           consumed++;
2810378                           if (!flag_star)
2810379                             {
2810380                               ptr_sshort = va_arg (ap, signed short *);
2810381                              *ptr_sshort = value_i;
2810382                               assigned++;
2810383                             }
```

```
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'o')
                      {
                        //------------------------ signed short, base 8.
                        value_i = strtointmax (input, &next, 8, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_sshort = va_arg (ap, signed short *);
                           *ptr_sshort = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'u')
                      {
                        //---------------------- unsigned short, base 10.
                        value_u = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ushort = va_arg (ap, unsigned short *);
                           *ptr_ushort = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'x' || format[f+2] == 'X')
                      {
                        //----------------------- signed short, base 16.
                        value_i = strtointmax (input, &next, 16, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_sshort = va_arg (ap, signed short *);
                           *ptr_sshort = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'n')
                      {
                        //----------- signed char, string index counter.
                        ptr_sshort = va_arg (ap, signed short *);
                       *ptr_sshort = (signed short)
                                     (input - start + scanned);
                        f += 2;
                      }
                    else
                      {
                        //------------ unsupported or unknown specifier.
                        f += 1;
                      }
                  }
                //------------------------- There is no 'long long int'.
                else if (format[f] == 'l')
                  {
                    //---------------------------------------- long int.
                    if      (format[f+1] == 'd')
                      {
                        //----------------------- signed long, base 10.
                        value_i = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_slong = va_arg (ap, signed long *);
                           *ptr_slong = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'i')
                      {
                        //------------------ signed long, base unknown.
                        value_i = strtointmax (input, &next, 0, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
```



```
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_slong = va_arg (ap, signed long *);
                           *ptr_slong = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'o')
                      {
                        //------------------------ signed long, base 8.
                        value_i = strtointmax (input, &next, 8, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_slong = va_arg (ap, signed long *);
                           *ptr_slong = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'u')
                      {
                        //---------------------- unsigned long, base 10.
                        value_u = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ulong = va_arg (ap, unsigned long *);
                           *ptr_ulong = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'x' || format[f+2] == 'X')
                      {
                        //----------------------- signed long, base 16.
                        value_i = strtointmax (input, &next, 16, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_slong = va_arg (ap, signed long *);
                           *ptr_slong = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'n')
                      {
                        //----------- signed char, string index counter.
                        ptr_slong = va_arg (ap, signed long *);
                       *ptr_slong = (signed long)
                                    (input - start + scanned);
                        f += 2;
                      }
                    else
                      {
                        //------------ unsupported or unknown specifier.
                        f += 1;
                      }
                  }
                else if (format[f] == 'j')
                  {
                    //---------------------------------------- intmax_t.
                    if      (format[f+1] == 'd')
                      {
                        //-------------------------- intmax_t, base 10.
                        value_i = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_simax = va_arg (ap, intmax_t *);
                           *ptr_simax = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'i')
                      {
```

```
                        //---------------------- intmax_t, base unknown.
                        value_i = strtointmax (input, &next, 0, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_simax = va_arg (ap, intmax_t *);
                           *ptr_simax = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'o')
                      {
                        //--------------------------- intmax_t, base 8.
                        value_i = strtointmax (input, &next, 8, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_simax = va_arg (ap, intmax_t *);
                           *ptr_simax = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'u')
                      {
                        //------------------------- uintmax_t, base 10.
                        value_u = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_uimax = va_arg (ap, uintmax_t *);
                           *ptr_uimax = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'x' || format[f+2] == 'X')
                      {
                        //--------------------------- intmax_t, base 16.
                        value_i = strtointmax (input, &next, 16, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_simax = va_arg (ap, intmax_t *);
                           *ptr_simax = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'n')
                      {
                        //---------- signed char, string index counter.
                        ptr_simax = va_arg (ap, intmax_t *);
                       *ptr_simax = (intmax_t)
                                    (input - start + scanned);
                        f += 2;
                      }
                    else
                      {
                        //------------ unsupported or unknown specifier.
                        f += 1;
                      }
                  }
                else if (format[f] == 'z')
                  {
                    //------------------------------------------ size_t.
                    if      (format[f+1] == 'd')
                      {
                        //---------------------------- size_t, base 10.
                        value_i = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_size = va_arg (ap, size_t *);
                           *ptr_size = value_i;
                            assigned++;
```



```
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'i')
                      {
                        //----------------------- size_t, base unknown.
                        value_i = strtointmax (input, &next, 0, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_size = va_arg (ap, size_t *);
                           *ptr_size = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'o')
                      {
                        //----------------------------- size_t, base 8.
                        value_i = strtointmax (input, &next, 8, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_size = va_arg (ap, size_t *);
                           *ptr_size = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'u')
                      {
                        //----------------------------- size_t, base 10.
                        value_u = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_size = va_arg (ap, size_t *);
                           *ptr_size = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'x' || format[f+2] == 'X')
                      {
                        //----------------------------- size_t, base 16.
                        value_i = strtointmax (input, &next, 16, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_size = va_arg (ap, size_t *);
                           *ptr_size = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'n')
                      {
                        //---------- signed char, string index counter.
                        ptr_size = va_arg (ap, size_t *);
                       *ptr_size = (size_t) (input - start + scanned);
                        f += 2;
                      }
                    else
                      {
                        //------------ unsupported or unknown specifier.
                        f += 1;
                      }
                  }
                else if (format[f] == 't')
                  {
                    //---------------------------------------- ptrdiff_t.
                    if      (format[f+1] == 'd')
                      {
                        //-------------------------- ptrdiff_t, base 10.
                        value_i = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
```

```
                        if (!flag_star)
                          {
                            ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                           *ptr_ptrdiff = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'i')
                      {
                        //---------------------- ptrdiff_t, base unknown.
                        value_i = strtointmax (input, &next, 0, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                           *ptr_ptrdiff = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'o')
                      {
                        //--------------------------- ptrdiff_t, base 8.
                        value_i = strtointmax (input, &next, 8, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                           *ptr_ptrdiff = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'u')
                      {
                        //------------------------- ptrdiff_t, base 10.
                        value_u = strtointmax (input, &next, 10, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                           *ptr_ptrdiff = value_u;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'x' || format[f+2] == 'X')
                      {
                        //------------------------- ptrdiff_t, base 16.
                        value_i = strtointmax (input, &next, 16, width);
                        if (input == next)
                          {
                            return (ass_or_eof (consumed, assigned));
                          }
                        consumed++;
                        if (!flag_star)
                          {
                            ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                           *ptr_ptrdiff = value_i;
                            assigned++;
                          }
                        f += 2;
                        input = next;
                      }
                    else if (format[f+1] == 'n')
                      {
                        //---------- signed char, string index counter.
                        ptr_ptrdiff = va_arg (ap, ptrdiff_t *);
                       *ptr_ptrdiff = (ptrdiff_t)
                                      (input - start + scanned);
                        f += 2;
                      }
                    else
                      {
                        //------------ unsupported or unknown specifier.
                        f += 1;
                      }
                  }
                //
                // Specifiers with no length modifier.
                //
                if      (format[f] == 'd')
                  {
                    //---------------------- signed short, base 10.
```



```
                    value_i = strtointmax (input, &next, 10, width);
                    if (input == next)
                      {
                        return (ass_or_eof (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sshort = va_arg (ap, signed short *);
                       *ptr_sshort = value_i;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'i')
                  {
                    //----------------------- signed int, base unknown.
                    value_i = strtointmax (input, &next, 0, width);
                    if (input == next)
                      {
                        return (ass_or_eof (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sint = va_arg (ap, signed int *);
                       *ptr_sint = value_i;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'o')
                  {
                    //---------------------------- signed int, base 8.
                    value_i = strtointmax (input, &next, 8, width);
                    if (input == next)
                      {
                        return (ass_or_eof (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sint = va_arg (ap, signed int *);
                       *ptr_sint = value_i;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'u')
                  {
                    //------------------------ unsigned short, base 10.
                    value_u = strtointmax (input, &next, 10, width);
                    if (input == next)
                      {
                        return (ass_or_eof (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_uint = va_arg (ap, unsigned int *);
                       *ptr_uint = value_u;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'x' || format[f] == 'X')
                  {
                    //-------------------------- signed short, base 16.
                    value_i = strtointmax (input, &next, 16, width);
                    if (input == next)
                      {
                        return (ass_or_eof (consumed, assigned));
                      }
                    consumed++;
                    if (!flag_star)
                      {
                        ptr_sint = va_arg (ap, signed int *);
                       *ptr_sint = value_i;
                        assigned++;
                      }
                    f += 1;
                    input = next;
                  }
                else if (format[f] == 'c')
                  {
                    //---------------------------------------- char[].
                    if (width == 0) width = 1;
                    //
                    if (!flag_star) ptr_char = va_arg (ap, char *);
                    //
                    for (count = 0;
                         width > 0 && *input != 0;
                         width--, ptr_char++, input++)
                      {
                        if (!flag_star) *ptr_char = *input;
                        //
                        count++;
```

```
2810990                       }
2810991                     //
2810992                     if (count)              consumed++;
2810993                     if (count && !flag_star) assigned++;
2810994                     //
2810995                     f += 1;
2810996                   }
2810997                 else if (format[f] == 's')
2810998                   {
2810999                     //---------------------------------------- string.
2811000                     if (!flag_star) ptr_char = va_arg (ap, char *);
2811001                     //
2811002                     for (count = 0;
2811003                          !isspace (*input) && *input != 0;
2811004                          ptr_char++, input++)
2811005                       {
2811006                         if (!flag_star) *ptr_char = *input;
2811007                         //
2811008                         count++;
2811009                       }
2811010                     if (!flag_star) *ptr_char = 0;
2811011                     //
2811012                     if (count)              consumed++;
2811013                     if (count && !flag_star) assigned++;
2811014                     //
2811015                     f += 1;
2811016                   }
2811017                 else if (format[f] == '[')
2811018                   {
2811019                     //
2811020                     f++;
2811021                     //
2811022                     if (format[f] == '^')
2811023                       {
2811024                         inverted = 1;
2811025                         f++;
2811026                       }
2811027                     else
2811028                       {
2811029                         inverted = 0;
2811030                       }
2811031                     //
2811032                     // Reset ascii array.
2811033                     //
2811034                     for (index = 0; index < 128; index++)
2811035                       {
2811036                         ascii[index] = inverted;
2811037                       }
2811038                     //
2811039                     //
2811040                     //
2811041                     for (count = 0;&format[f] < end_format; count++)
2811042                       {
2811043                         if (format[f] == ']' && count > 0)
2811044                           {
2811045                             break;
2811046                           }
2811047                         //
2811048                         // Check for an interval.
2811049                         //
2811050                         if (format[f+1] == '-'
2811051                          && format[f+2] != ']'
2811052                          && format[f+2] != 0)
2811053                           {
2811054                             //
2811055                             // Interval.
2811056                             //
2811057                             for (index = format[f];
2811058                                  index <= format[f+2];
2811059                                  index++)
2811060                               {
2811061                                 ascii[index] = !inverted;
2811062                               }
2811063                             f += 3;
2811064                             continue;
2811065                           }
2811066                         //
2811067                         // Single character.
2811068                         //
2811069                         index = format[f];
2811070                         ascii[index] = !inverted;
2811071                         f++;
2811072                       }
2811073                     //
2811074                     // Is the scan correctly finished?.
2811075                     //
2811076                     if (format[f] != ']')
2811077                       {
2811078                         return (ass_or_eof (consumed, assigned));
2811079                       }
2811080                     //
2811081                     // The ascii table is populated.
2811082                     //
2811083                     if (width == 0) width = SIZE_MAX;
2811084                     //
2811085                     // Scan the input string.
2811086                     //
2811087                     if (!flag_star) ptr_char = va_arg (ap, char *);
2811088                     //
2811089                     for (count = 0;
2811090                          width > 0 && *input != 0;
```



```
2811091                          width--, ptr_char++, input++)
2811092                       {
2811093                         index = *input;
2811094                         if (ascii[index])
2811095                           {
2811096                             if (!flag_star) *ptr_char = *input;
2811097                             count++;
2811098                           }
2811099                         else
2811100                           {
2811101                             break;
2811102                           }
2811103                       }
2811104                     //
2811105                     if (count)              consumed++;
2811106                     if (count && !flag_star) assigned++;
2811107                     //
2811108                     f += 1;
2811109                   }
2811110                 else if (format[f] == 'p')
2811111                   {
2811112                     //---------------------------------------- void *.
2811113                     value_i = strtointmax (input, &next, 16, width);
2811114                     if (input == next)
2811115                       {
2811116                         return (ass_or_eof (consumed, assigned));
2811117                       }
2811118                     consumed++;
2811119                     if (!flag_star)
2811120                       {
2811121                         ptr_void = va_arg (ap, void **);
2811122                        *ptr_void = (void *) ((int) value_i);
2811123                         assigned++;
2811124                       }
2811125                     f += 1;
2811126                     input = next;
2811127                   }
2811128                 else if (format[f] == 'n')
2811129                   {
2811130                     //--------------- signed char, string index counter.
2811131                     ptr_sint = va_arg (ap, signed int *);
2811132                     *ptr_sint = (signed char) (input - start + scanned);
2811133                     f += 1;
2811134                   }
2811135                 else
2811136                   {
2811137                     //--------------- unsupported or unknown specifier.
2811138                     ;
2811139                   }
2811140
2811141                 //-----------------------------------------------------
2811142                 // End of specifier.
2811143                 //-----------------------------------------------------
2811144
2811145                 width_string[0]     = '\0';
2811146                 specifier           = 0;
2811147                 specifier_flags     = 0;
2811148                 specifier_width     = 0;
2811149                 specifier_type      = 0;
2811150                 flag_star           = 0;
2811151
2811152               }
2811153             }
2811154           //
2811155           // The format or the input string is terminated.
2811156           //
2811157           if  (&format[f] < end_format && stream)
2811158             {
2811159               //
2811160               // Only the input string is finished, and the input comes
2811161               // from a stream, so another read will be done.
2811162               //
2811163               scanned += (int) (input - start);
2811164               continue;
2811165             }
2811166           //
2811167           // The format string is terminated.
2811168           //
2811169           return (ass_or_eof (consumed, assigned));
2811170         }
2811171 }
2811172 //---------------------------------------------------------------------
2811173 static intmax_t
2811174 strtointmax (const char *restrict string, char **endptr,
2811175              int base, size_t max_width)
2811176 {
2811177     int      i;
2811178     int      d;                 // Digits counter.
2811179     int      sign = +1;
2811180     intmax_t number;
2811181     intmax_t previous;
2811182     int      digit;
2811183     //
2811184     bool     flag_prefix_oct = 0;
2811185     bool     flag_prefix_exa = 0;
2811186     bool     flag_prefix_dec = 0;
2811187     //
2811188     // If the 'max_width' value is zero, fix it to the maximum
2811189     // that it can represent.
2811190     //
2811191     if (max_width == 0)
```

```
2811192       {
2811193         max_width = SIZE_MAX;
2811194       }
2811195     //
2811196     // Eat initial spaces, but if there are spaces, there is an
2811197     // error inside the calling function!
2811198     //
2811199     for (i = 0; isspace (string[i]); i++)
2811200       {
2811201         fprintf (stderr, "libc error: file \"%s\", line %i\n",
2811202                  __FILE__, __LINE__);
2811203         ;
2811204       }
2811205     //
2811206     // Check sign. The ‘max_width’ counts also the sign, if there is
2811207     // one.
2811208     //
2811209     if (string[i] == '+')
2811210       {
2811211         sign = +1;
2811212         i++;
2811213         max_width--;
2811214       }
2811215     else if (string[i] == '-')
2811216       {
2811217         sign = -1;
2811218         i++;
2811219         max_width--;
2811220       }
2811221     //
2811222     // Check for prefix.
2811223     //
2811224     if (string[i] == '0')
2811225       {
2811226         if (string[i+1] == 'x' || string[i+1] == 'X')
2811227           {
2811228             flag_prefix_exa = 1;
2811229           }
2811230         if (isdigit (string[i+1]))
2811231           {
2811232             flag_prefix_oct = 1;
2811233           }
2811234       }
2811235     //
2811236     if (string[i] > '0' && string[i] <= '9')
2811237       {
2811238         flag_prefix_dec = 1;
2811239       }
2811240     //
2811241     // Check compatibility with requested base.
2811242     //
2811243     if (flag_prefix_exa)
2811244       {
2811245         if (base == 0)
2811246           {
2811247             base = 16;
2811248           }
2811249         else if (base == 16)
2811250           {
2811251             ;   // Ok.
2811252           }
2811253         else
2811254           {
2811255             //
2811256             // Incompatible sequence: only the initial zero is reported.
2811257             //
2811258             *endptr = &string[i+1];
2811259             return ((intmax_t) 0);
2811260           }
2811261         //
2811262         // Move on, after the ‘0x’ prefix.
2811263         //
2811264         i += 2;
2811265       }
2811266     //
2811267     if (flag_prefix_oct)
2811268       {
2811269         if (base == 0)
2811270           {
2811271             base = 8;
2811272           }
2811273         //
2811274         // Move on, after the ‘0’ prefix.
2811275         //
2811276         i += 1;
2811277       }
2811278     //
2811279     if (flag_prefix_dec)
2811280       {
2811281         if (base == 0)
2811282           {
2811283             base = 10;
2811284           }
2811285       }
2811286     //
2811287     // Scan the string.
2811288     //
2811289     for (d = 0, number = 0; d < max_width && string[i] != 0; i++, d++)
2811290       {
2811291         if      (string[i] >= '0' && string[i] <= '9')
2811292           {
```



```
2811293             digit = string[i] - '0';
2811294           }
2811295         else if (string[i] >= 'A' && string[i] <= 'F')
2811296           {
2811297             digit = string[i] - 'A' + 10;
2811298           }
2811299         else if (string[i] >= 'a' && string[i] <= 'f')
2811300           {
2811301             digit = string[i] - 'a' + 10;
2811302           }
2811303         else
2811304           {
2811305             digit = 999;
2811306           }
2811307         //
2811308         // Give a sign to the digit.
2811309         //
2811310         digit *= sign;
2811311         //
2811312         // Compare with the base.
2811313         //
2811314         if (base > (digit * sign))
2811315           {
2811316             //
2811317             // Check if the current digit can be safely computed.
2811318             //
2811319             previous = number;
2811320             number  *= base;
2811321             number  += digit;
2811322             if (number / base != previous)
2811323               {
2811324                 //
2811325                 // Out of range.
2811326                 //
2811327                 *endptr = &string[i+1];
2811328                 errset (ERANGE);        // Result too large.
2811329                 if (sign > 0)
2811330                   {
2811331                     return (INTMAX_MAX);
2811332                   }
2811333                 else
2811334                   {
2811335                     return (INTMAX_MIN);
2811336                   }
2811337               }
2811338           }
2811339         else
2811340           {
2811341             *endptr = &string[i];
2811342             return (number);
2811343           }
2811344       }
2811345     //
2811346     // The string is finished or the max digits length is reached.
2811347     //
2811348     *endptr = &string[i];
2811349     //
2811350     return (number);
2811351 }
2811352 //---------------------------------------------------------------------
2811353 static int
2811354 ass_or_eof (int consumed, int assigned)
2811355 {
2811356     if (consumed == 0)
2811357       {
2811358         return (EOF);
2811359       }
2811360     else
2811361       {
2811362         return (assigned);
2811363       }
2811364 }
2811365 //---------------------------------------------------------------------
```

lib/stdio/vprintf.c

Si veda la sezione u0.128.

```
2820001 #include <stdio.h>
2820002 #include <sys/types.h>
2820003 #include <sys/os16.h>
2820004 #include <string.h>
2820005 #include <unistd.h>
2820006 //---------------------------------------------------------------------
2820007 int
2820008 vprintf (char *restrict format, va_list arg)
2820009 {
2820010     ssize_t     size_written;
2820011     size_t      size;
2820012     size_t      size_total;
2820013     int         status;
2820014     char        string[BUFSIZ];
2820015     char       *buffer = string;
2820016
2820017     buffer[0] = 0;
2820018     status = vsprintf (buffer, format, arg);
2820019
2820020     size = strlen (buffer);
2820021     if (size >= BUFSIZ)
2820022       {
2820023         size = BUFSIZ;
```

```
2820024       }
2820025
2820026     for (size_total = 0, size_written = 0;
2820027          size_total < size;
2820028          size_total += size_written, buffer += size_written)
2820029       {
2820030         //
2820031         // Write to the standard output: file descriptor n. 1.
2820032         //
2820033         size_written = write (STDOUT_FILENO, buffer, size - size_total);
2820034         if (size_written < 0)
2820035           {
2820036             return (size_total);
2820037           }
2820038       }
2820039     return (size);
2820040 }
```

lib/stdio/vscanf.c

«

Si veda la sezione u0.129.

```
2830001 #include <stdio.h>
2830002 //---------------------------------------------------------------------
2830003 int
2830004 vscanf (const char *restrict format, va_list ap)
2830005 {
2830006     return (vfscanf (stdin, format, ap));
2830007 }
2830008 //---------------------------------------------------------------------
```

lib/stdio/vsnprintf.c

«

Si veda la sezione u0.128.

```
2840001 #include <stdint.h>
2840002 #include <stdbool.h>
2840003 #include <stdlib.h>
2840004 #include <string.h>
2840005 #include <stdio.h>
2840006 //---------------------------------------------------------------------
2840007 static size_t uimaxtoa      (uintmax_t integer, char *buffer, int base,
2840008                              int uppercase, size_t size);
2840009 static size_t imaxtoa       (intmax_t  integer, char *buffer, int base,
2840010                              int uppercase, size_t size);
2840011 static size_t simaxtoa      (intmax_t  integer, char *buffer, int base,
2840012                              int uppercase, size_t size);
2840013 static size_t uimaxtoa_fill (uintmax_t integer, char *buffer, int base,
2840014                              int uppercase, int width, int filler,
2840015                              int max);
2840016 static size_t imaxtoa_fill  (intmax_t  integer, char *buffer, int base,
2840017                              int uppercase, int width, int filler,
2840018                              int max);
2840019 static size_t simaxtoa_fill (intmax_t  integer, char *buffer, int base,
2840020                              int uppercase, int width, int filler,
2840021                              int max);
2840022 static size_t strtostr_fill (char *string, char *buffer, int width,
2840023                              int filler, int max);
2840024 //---------------------------------------------------------------------
2840025 int
2840026 vsnprintf (char *restrict string, size_t size,
2840027            const char *restrict format, va_list ap)
2840028 {
2840029     //
2840030     // We produce at most 'size-1' characters, + '\0'.
2840031     // 'size' is used also as the max size for internal
2840032     // strings, but only if it is not too big.
2840033     //
2840034     int        f                   = 0;
2840035     int        s                   = 0;
2840036     int        remain              = size - 1;
2840037     //
2840038     bool       specifier           = 0;
2840039     bool       specifier_flags     = 0;
2840040     bool       specifier_width     = 0;
2840041     bool       specifier_precision = 0;
2840042     bool       specifier_type      = 0;
2840043     //
2840044     bool       flag_plus           = 0;
2840045     bool       flag_minus          = 0;
2840046     bool       flag_space          = 0;
2840047     bool       flag_alternate      = 0;
2840048     bool       flag_zero           = 0;
2840049     //
2840050     int        alignment;
2840051     int        filler;
2840052     //
2840053     intmax_t   value_i;
2840054     uintmax_t  value_ui;
2840055     char      *value_cp;
2840056     //
2840057     size_t     width;
2840058     size_t     precision;
2840059     #define    str_size  BUFSIZ/2
2840060     char       width_string[str_size];
2840061     char       precision_string[str_size];
2840062     int        w;
2840063     int        p;
2840064     //
2840065     width_string[0]     = '\0';
```



```
2840066     precision_string[0] = '\0';
2840067     //
2840068     while (format[f] != 0 && s < (size - 1))
2840069       {
2840070         if (!specifier)
2840071           {
2840072             //------------------- The context is not inside a specifier.
2840073             if (format[f] != '%')
2840074               {
2840075                 string[s] = format[f];
2840076                 s++;
2840077                 remain--;
2840078                 f++;
2840079                 continue;
2840080               }
2840081             if (format[f] == '%' && format[f+1] == '%')
2840082               {
2840083                 string[s] = '%';
2840084                 f++;
2840085                 f++;
2840086                 s++;
2840087                 remain--;
2840088                 continue;
2840089               }
2840090             if (format[f] == '%')
2840091               {
2840092                 f++;
2840093                 specifier = 1;
2840094                 specifier_flags = 1;
2840095                 continue;
2840096               }
2840097           }
2840098         //
2840099         if (specifier && specifier_flags)
2840100           {
2840101             //------------------- The context is inside specifier flags.
2840102             if (format[f] == '+')
2840103               {
2840104                 flag_plus = 1;
2840105                 f++;
2840106                 continue;
2840107               }
2840108             else if (format[f] == '-')
2840109               {
2840110                 flag_minus = 1;
2840111                 f++;
2840112                 continue;
2840113               }
2840114             else if (format[f] == ' ')
2840115               {
2840116                 flag_space = 1;
2840117                 f++;
2840118                 continue;
2840119               }
2840120             else if (format[f] == '#')
2840121               {
2840122                 flag_alternate = 1;
2840123                 f++;
2840124                 continue;
2840125               }
2840126             else if (format[f] == '0')
2840127               {
2840128                 flag_zero = 1;
2840129                 f++;
2840130                 continue;
2840131               }
2840132             else
2840133               {
2840134                 specifier_flags = 0;
2840135                 specifier_width = 1;
2840136               }
2840137           }
2840138         //
2840139         if (specifier && specifier_width)
2840140           {
2840141             //------------------- The context is inside specifier width.
2840142             for (w = 0; format[f] >= '0' && format[f] <= '9'
2840143                  && w < str_size; w++)
2840144               {
2840145                 width_string[w] = format[f];
2840146                 f++;
2840147               }
2840148             width_string[w] = '\0';
2840149
2840150             specifier_width = 0;
2840151
2840152             if (format[f] == '.')
2840153               {
2840154                 specifier_precision = 1;
2840155                 f++;
2840156               }
2840157             else
2840158               {
2840159                 specifier_precision = 0;
2840160                 specifier_type      = 1;
2840161               }
2840162           }
2840163         //
2840164         if (specifier && specifier_precision)
2840165           {
2840166             //--------------- The context is inside specifier precision.
```

```
2840167             for (p = 0; format[f] >= '0' && format[f] <= '9'
2840168                  && p < str_size; p++)
2840169               {
2840170                 precision_string[p] = format[f];
2840171                 p++;
2840172               }
2840173             precision_string[p] = '\0';
2840174
2840175             specifier_precision = 0;
2840176             specifier_type      = 1;
2840177           }
2840178         //
2840179         if (specifier && specifier_type)
2840180           {
2840181             //-------------------- The context is inside specifier type.
2840182             width     = atoi (width_string);
2840183             precision = atoi (precision_string);
2840184                         filler = ' ';
2840185             if (flag_zero)  filler = '0';
2840186             if (flag_space) filler = ' ';
2840187                         alignment = width;
2840188             if (flag_minus)
2840189               {
2840190                 alignment = -alignment;
2840191                 filler = ' ';   // The filler character cannot
2840192                                 // be zero, so it is black.
2840193               }
2840194             //
2840195             if (format[f] == 'h' && format[f+1] == 'h')
2840196               {
2840197                 if (format[f+2] == 'd' || format[f+2] == 'i')
2840198                   {
2840199                     //---------------------------- signed char, base 10.
2840200                     value_i = va_arg (ap, int);
2840201                     if (flag_plus)
2840202                       {
2840203                         s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840204                                             alignment, filler, remain);
2840205                       }
2840206                     else
2840207                       {
2840208                         s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840209                                            alignment, filler, remain);
2840210                       }
2840211                     f += 3;
2840212                   }
2840213                 else if (format[f+2] == 'u')
2840214                   {
2840215                     //------------------------- unsigned char, base 10.
2840216                     value_ui = va_arg (ap, unsigned int);
2840217                     s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840218                                         alignment, filler, remain);
2840219                     f += 3;
2840220                   }
2840221                 else if (format[f+2] == 'o')
2840222                   {
2840223                     //------------------------- unsigned char, base 8.
2840224                     value_ui = va_arg (ap, unsigned int);
2840225                     s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840226                                         alignment, filler, remain);
2840227                     f += 3;
2840228                   }
2840229                 else if (format[f+2] == 'x')
2840230                   {
2840231                     //------------------------- unsigned char, base 16.
2840232                     value_ui = va_arg (ap, unsigned int);
2840233                     s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840234                                         alignment, filler, remain);
2840235                     f += 3;
2840236                   }
2840237                 else if (format[f+2] == 'X')
2840238                   {
2840239                     //------------------------- unsigned char, base 16.
2840240                     value_ui = va_arg (ap, unsigned int);
2840241                     s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840242                                         alignment, filler, remain);
2840243                     f += 3;
2840244                   }
2840245                 else
2840246                   {
2840247                     //---------------- unsupported or unknown specifier.
2840248                     f += 2;
2840249                   }
2840250               }
2840251             else if (format[f] == 'h')
2840252               {
2840253                 if (format[f+1] == 'd' || format[f+1] == 'i')
2840254                   {
2840255                     //------------------------------ short int, base 10.
2840256                     value_i = va_arg (ap, int);
2840257                     if (flag_plus)
2840258                       {
2840259                         s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840260                                             alignment, filler, remain);
2840261                       }
2840262                     else
2840263                       {
2840264                         s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840265                                            alignment, filler, remain);
2840266                       }
2840267                     f += 2;
```



```
2840268                   }
2840269                 else if (format[f+1] == 'u')
2840270                   {
2840271                     //---------------------- unsigned short int, base 10.
2840272                     value_ui = va_arg (ap, unsigned int);
2840273                     s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840274                                         alignment, filler, remain);
2840275                     f += 2;
2840276                   }
2840277                 else if (format[f+1] == 'o')
2840278                   {
2840279                     //---------------------- unsigned short int, base 8.
2840280                     value_ui = va_arg (ap, unsigned int);
2840281                     s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840282                                         alignment, filler, remain);
2840283                     f += 2;
2840284                   }
2840285                 else if (format[f+1] == 'x')
2840286                   {
2840287                     //---------------------- unsigned short int, base 16.
2840288                     value_ui = va_arg (ap, unsigned int);
2840289                     s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840290                                         alignment, filler, remain);
2840291                     f += 2;
2840292                   }
2840293                 else if (format[f+1] == 'X')
2840294                   {
2840295                     //---------------------- unsigned short int, base 16.
2840296                     value_ui = va_arg (ap, unsigned int);
2840297                     s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840298                                         alignment, filler, remain);
2840299                     f += 2;
2840300                   }
2840301                 else
2840302                   {
2840303                     //---------------- unsupported or unknown specifier.
2840304                     f += 1;
2840305                   }
2840306               }
2840307
2840308             //-----------------------------------------------------
2840309             // There is no 'long long int'.
2840310             //-----------------------------------------------------
2840311
2840312           else if (format[f] == 'l')
2840313             {
2840314               if (format[f+1] == 'd' || format[f+1] == 'i')
2840315                 {
2840316                   //------------------------------ long int base 10.
2840317                   value_i = va_arg (ap, long int);
2840318                   if (flag_plus)
2840319                     {
2840320                       s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840321                                           alignment, filler, remain);
2840322                     }
2840323                   else
2840324                     {
2840325                       s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840326                                          alignment, filler, remain);
2840327                     }
2840328                   f += 2;
2840329                 }
2840330               else if (format[f+1] == 'u')
2840331                 {
2840332                   //----------------------- Unsigned long int base 10.
2840333                   value_ui = va_arg (ap, unsigned long int);
2840334                   s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840335                                       alignment, filler, remain);
2840336                   f += 2;
2840337                 }
2840338               else if (format[f+1] == 'o')
2840339                 {
2840340                   //----------------------- Unsigned long int base 8.
2840341                   value_ui = va_arg (ap, unsigned long int);
2840342                   s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840343                                       alignment, filler, remain);
2840344                   f += 2;
2840345                 }
2840346               else if (format[f+1] == 'x')
2840347                 {
2840348                   //----------------------- Unsigned long int base 16.
2840349                   value_ui = va_arg (ap, unsigned long int);
2840350                   s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840351                                       alignment, filler, remain);
2840352                   f += 2;
2840353                 }
2840354               else if (format[f+1] == 'X')
2840355                 {
2840356                   //----------------------- Unsigned long int base 16.
2840357                   value_ui = va_arg (ap, unsigned long int);
2840358                   s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840359                                       alignment, filler, remain);
2840360                   f += 2;
2840361                 }
2840362               else
2840363                 {
2840364                   //---------------- unsupported or unknown specifier.
2840365                   f += 1;
2840366                 }
2840367             }
2840368           else if (format[f] == 'j')
```

```
2840369                 {
2840370                   if (format[f+1] == 'd' || format[f+1] == 'i')
2840371                     {
2840372                       //------------------------------- intmax_t base 10.
2840373                       value_i = va_arg (ap, intmax_t);
2840374                       if (flag_plus)
2840375                         {
2840376                           s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840377                                               alignment, filler, remain);
2840378                         }
2840379                       else
2840380                         {
2840381                           s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840382                                              alignment, filler, remain);
2840383                         }
2840384                       f += 2;
2840385                     }
2840386                   else if (format[f+1] == 'u')
2840387                     {
2840388                       //------------------------------- uintmax_t base 10.
2840389                       value_ui = va_arg (ap, uintmax_t);
2840390                       s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840391                                           alignment, filler, remain);
2840392                       f += 2;
2840393                     }
2840394                   else if (format[f+1] == 'o')
2840395                     {
2840396                       //------------------------------- uintmax_t base 8.
2840397                       value_ui = va_arg (ap, uintmax_t);
2840398                       s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840399                                           alignment, filler, remain);
2840400                       f += 2;
2840401                     }
2840402                   else if (format[f+1] == 'x')
2840403                     {
2840404                       //------------------------------- uintmax_t base 16.
2840405                       value_ui = va_arg (ap, uintmax_t);
2840406                       s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840407                                           alignment, filler, remain);
2840408                       f += 2;
2840409                     }
2840410                   else if (format[f+1] == 'X')
2840411                     {
2840412                       //------------------------------- uintmax_t base 16.
2840413                       value_ui = va_arg (ap, uintmax_t);
2840414                       s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840415                                           alignment, filler, remain);
2840416                       f += 2;
2840417                     }
2840418                   else
2840419                     {
2840420                       //---------------- unsupported or unknown specifier.
2840421                       f += 1;
2840422                     }
2840423                 }
2840424               else if (format[f] == 'z')
2840425                 {
2840426                   if (format[f+1] == 'd'
2840427                       || format[f+1] == 'i'
2840428                       || format[f+1] == 'i')
2840429                     {
2840430                       //-------------------------------- size_t base 10.
2840431                       value_ui = va_arg (ap, unsigned long int);
2840432                       s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840433                                           alignment, filler, remain);
2840434                       f += 2;
2840435                     }
2840436                   else if (format[f+1] == 'o')
2840437                     {
2840438                       //-------------------------------- size_t base 8.
2840439                       value_ui = va_arg (ap, unsigned long int);
2840440                       s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840441                                           alignment, filler, remain);
2840442                       f += 2;
2840443                     }
2840444                   else if (format[f+1] == 'x')
2840445                     {
2840446                       //-------------------------------- size_t base 16.
2840447                       value_ui = va_arg (ap, unsigned long int);
2840448                       s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840449                                           alignment, filler, remain);
2840450                       f += 2;
2840451                     }
2840452                   else if (format[f+1] == 'X')
2840453                     {
2840454                       //-------------------------------- size_t base 16.
2840455                       value_ui = va_arg (ap, unsigned long int);
2840456                       s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840457                                           alignment, filler, remain);
2840458                       f += 2;
2840459                     }
2840460                   else
2840461                     {
2840462                       //---------------- unsupported or unknown specifier.
2840463                       f += 1;
2840464                     }
2840465                 }
2840466               else if (format[f] == 't')
2840467                 {
2840468                   if (format[f+1] == 'd' || format[f+1] == 'i')
2840469                     {
```



```
2840470                       //------------------------------- ptrdiff_t base 10.
2840471                       value_i = va_arg (ap, long int);
2840472                       if (flag_plus)
2840473                         {
2840474                           s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840475                                               alignment, filler, remain);
2840476                         }
2840477                       else
2840478                         {
2840479                           s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840480                                              alignment, filler, remain);
2840481                         }
2840482                       f += 2;
2840483                     }
2840484                   else if (format[f+1] == 'u')
2840485                     {
2840486                       //----------------- ptrdiff_t base 10, without sign.
2840487                       value_ui = va_arg (ap, unsigned long int);
2840488                       s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840489                                           alignment, filler, remain);
2840490                       f += 2;
2840491                     }
2840492                   else if (format[f+1] == 'o')
2840493                     {
2840494                       //----------------- ptrdiff_t base 8, without sign.
2840495                       value_ui = va_arg (ap, unsigned long int);
2840496                       s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840497                                           alignment, filler, remain);
2840498                       f += 2;
2840499                     }
2840500                   else if (format[f+1] == 'x')
2840501                     {
2840502                       //----------------- ptrdiff_t base 16, without sign.
2840503                       value_ui = va_arg (ap, unsigned long int);
2840504                       s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840505                                           alignment, filler, remain);
2840506                       f += 2;
2840507                     }
2840508                   else if (format[f+1] == 'X')
2840509                     {
2840510                       //----------------- ptrdiff_t base 16, without sign.
2840511                       value_ui = va_arg (ap, unsigned long int);
2840512                       s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840513                                           alignment, filler, remain);
2840514                       f += 2;
2840515                     }
2840516                   else
2840517                     {
2840518                       //---------------- unsupported or unknown specifier.
2840519                       f += 1;
2840520                     }
2840521                 }
2840522               if (format[f] == 'd' || format[f] == 'i')
2840523                 {
2840524                   //---------------------------------------- int base 10.
2840525                   value_i = va_arg (ap, int);
2840526                   if (flag_plus)
2840527                     {
2840528                       s += simaxtoa_fill (value_i, &string[s], 10, 0,
2840529                                           alignment, filler, remain);
2840530                     }
2840531                   else
2840532                     {
2840533                       s += imaxtoa_fill (value_i, &string[s], 10, 0,
2840534                                          alignment, filler, remain);
2840535                     }
2840536                   f += 1;
2840537                 }
2840538               else if (format[f] == 'u')
2840539                 {
2840540                   //------------------------------- unsigned int base 10.
2840541                   value_ui = va_arg (ap, unsigned int);
2840542                   s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
2840543                                       alignment, filler, remain);
2840544                   f += 1;
2840545                 }
2840546               else if (format[f] == 'o')
2840547                 {
2840548                   //------------------------------- unsigned int base 8.
2840549                   value_ui = va_arg (ap, unsigned int);
2840550                   s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
2840551                                       alignment, filler, remain);
2840552                   f += 1;
2840553                 }
2840554               else if (format[f] == 'x')
2840555                 {
2840556                   //------------------------------- unsigned int base 16.
2840557                   value_ui = va_arg (ap, unsigned int);
2840558                   s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
2840559                                       alignment, filler, remain);
2840560                   f += 1;
2840561                 }
2840562               else if (format[f] == 'X')
2840563                 {
2840564                   //------------------------------- unsigned int base 16.
2840565                   value_ui = va_arg (ap, unsigned int);
2840566                   s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
2840567                                       alignment, filler, remain);
2840568                   f += 1;
2840569                 }
2840570               else if (format[f] == 'c')
```

```
              {
                //---------------------------------------- unsigned char.
                value_ui = va_arg (ap, unsigned int);
                string[s] = (char) value_ui;
                s += 1;
                f += 1;
              }
            else if (format[f] == 's')
              {
                //------------------------------------------------ string.
                value_cp = va_arg (ap, char *);
                filler = ' ';

                s += strtostr_fill (value_cp, &string[s], alignment,
                                    filler, remain);
                f += 1;
              }
            else
              {
                //-------------------- unsupported or unknown specifier.
                ;
              }
            //-----------------------------------------------------------
            // End of specifier.
            //-----------------------------------------------------------
            width_string[0]     = '\0';
            precision_string[0] = '\0';

            specifier           = 0;
            specifier_flags     = 0;
            specifier_width     = 0;
            specifier_precision = 0;
            specifier_type      = 0;

            flag_plus           = 0;
            flag_minus          = 0;
            flag_space          = 0;
            flag_alternate      = 0;
            flag_zero           = 0;
          }
      }
    string[s] = '\0';
    return s;
}
//---------------------------------------------------------------------
// Static functions.
//---------------------------------------------------------------------
static size_t
uimaxtoa (uintmax_t integer, char *buffer, int base, int uppercase,
          size_t size)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer into a string.
    //-----------------------------------------------------------------

        uintmax_t  integer_copy = integer;
           size_t  digits;
              int  b;
    unsigned char  remainder;

    for (digits = 0; integer_copy > 0; digits++)
      {
        integer_copy = integer_copy / base;
      }

    if (buffer == NULL && integer == 0) return 1;
    if (buffer == NULL && integer > 0)  return digits;

    if (integer == 0)
      {
        buffer[0] = '0';
        buffer[1] = '\0';
        return 1;
      }
    //
    // Fix the maximum number of digits.
    //
    if (size > 0 && digits > size) digits = size;
    //
    *(buffer + digits) = '\0';            // End of string.

    for (b = digits - 1; integer != 0 && b >= 0; b--)
      {
        remainder = integer % base;
        integer   = integer / base;

        if (remainder <= 9)
          {
            *(buffer + b) = remainder + '0';
          }
        else
          {
            if (uppercase)
              {
                *(buffer + b) = remainder - 10 + 'A';
              }
            else
              {
                *(buffer + b) = remainder - 10 + 'a';
              }
          }
```



```
      }
    return digits;
}
//---------------------------------------------------------------------
static size_t
imaxtoa (intmax_t integer, char *buffer, int base, int uppercase,
         size_t size)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer with sign into a string.
    //-----------------------------------------------------------------

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, size);
      }
    //
    // At this point, there is a negative number, less than zero.
    //
    if (buffer == NULL)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, size) + 1;
      }

    *buffer = '-';        // The minus sign is needed at the beginning.
    if (size == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, size-1)
               + 1;
      }
}
//---------------------------------------------------------------------
static size_t
simaxtoa (intmax_t integer, char *buffer, int base, int uppercase,
          size_t size)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer with sign into a string, placing
    // the sign also if it is positive.
    //-----------------------------------------------------------------

    if (buffer == NULL && integer >= 0)
      {
        return uimaxtoa (integer, NULL, base, uppercase, size) + 1;
      }

    if (buffer == NULL && integer < 0)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, size) + 1;
      }
    //
    // At this point, 'buffer' is different from NULL.
    //
    if (integer >= 0)
      {
        *buffer = '+';
      }
    else
      {
        *buffer = '-';
      }

    if (size == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer+1, base, uppercase, size-1)
               + 1;
      }
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, size-1)
               + 1;
      }
}
//---------------------------------------------------------------------
static size_t
uimaxtoa_fill (uintmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer without sign into a string,
    // takeing care of the alignment.
    //-----------------------------------------------------------------

    size_t size_i;
    size_t size_f;

    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_i = uimaxtoa (integer, NULL, base, uppercase, 0);
```

```
    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Left alignment.
        uimaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Right alignment.
        memset (buffer, filler, size_f);
        uimaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//---------------------------------------------------------------------
static size_t
imaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer with sign into a string,
    // takeing care of the alignment.
    //-----------------------------------------------------------------

    size_t size_i;
    size_t size_f;

    if (max < 0) return 0; // 'max' must be a positive value.

    size_i = imaxtoa (integer, NULL, base, uppercase, 0);

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    // size_i <= abs (width).

    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Left alignment.
        imaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Right alignment.
        memset (buffer, filler, size_f);
        imaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//---------------------------------------------------------------------
static size_t
simaxtoa_fill (intmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    //-----------------------------------------------------------------
    // Convert a maximum rank integer with sign into a string,
    // placing the sign also if it is positive and takeing care of the
```



```
    // alignment.
    //-----------------------------------------------------------------

    size_t size_i;
    size_t size_f;

    if (max < 0) return 0; // 'max' must be a positive value.

    size_i = simaxtoa (integer, NULL, base, uppercase, 0);

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return simaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Left alignment.
        simaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Right alignment.
        memset (buffer, filler, size_f);
        simaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//---------------------------------------------------------------------
static size_t
strtostr_fill (char *string, char *buffer, int width, int filler,
               int max)
{
    //-----------------------------------------------------------------
    // Transfer a string with care for the alignment.
    //-----------------------------------------------------------------

    size_t size_s;
    size_t size_f;

    if (max < 0) return 0; // 'max' must be a positive value.

    size_s = strlen (string);

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (width != 0 && size_s > abs (width))
      {
        memcpy (buffer, string, abs (width));
        buffer[width] = '\0';
        return width;
      }

    if (width == 0 && max > 0 && size_s > max)
      {
        memcpy (buffer, string, max);
        buffer[max] = '\0';
        return max;
      }

    if (width == 0 && max > 0 && size_s < max)
      {
        memcpy (buffer, string, size_s);
        buffer[size_s] = '\0';
        return size_s;
      }
    //
    // width =! 0
    // size_s <= abs (width)
    //
    size_f = abs (width) - size_s;

    if (width < 0)
      {
        // Right alignment.
        memset (buffer, filler, size_f);
        strncpy (buffer+size_f, string, size_s);
      }
    else
      {
```

```
2840975         // Left alignment.
2840976         strncpy (buffer, string, size_s);
2840977         memset (buffer+size_s, filler, size_f);
2840978       }
2840979     *(buffer + abs (width)) = '\0';
2840980
2840981     return abs (width);
2840982 }
2840983
2840984
```

lib/stdio/vsprintf.c

«

Si veda la sezione u0.128.

```
2850001 #include <stdio.h>
2850002 #include <sys/os16.h>
2850003 //-----------------------------------------------------------------------
2850004 int
2850005 vsprintf (char *string, char *restrict format, va_list arg)
2850006 {
2850007     return (vsnprintf (string, BUFSIZ, format, arg));
2850008 }
```

lib/stdio/vsscanf.c

«

Si veda la sezione u0.129.

```
2860001 #include <stdio.h>
2860002
2860003 //-----------------------------------------------------------------------
2860004 int vfsscanf (FILE *restrict fp, const char *string,
2860005               const char *restrict format, va_list ap);
2860006 //-----------------------------------------------------------------------
2860007 int
2860008 vsscanf (const char *string, const char *restrict format, va_list ap)
2860009 {
2860010     return (vfsscanf (NULL, string, format, ap));
2860011 }
2860012 //-----------------------------------------------------------------------
```

os16: «lib/stdlib.h»

«

Si veda la sezione u0.2.

```
2870001 #ifndef _STDLIB_H
2870002 #define _STDLIB_H       1
2870003 //-----------------------------------------------------------------------
2870004
2870005 #include <size_t.h>
2870006 #include <wchar_t.h>
2870007 #include <NULL.h>
2870008 #include <limits.h>
2870009 #include <const.h>
2870010 #include <restrict.h>
2870011 //-----------------------------------------------------------------------
2870012 typedef struct {
2870013     int      quot;
2870014     int      rem;
2870015 } div_t;
2870016 //-----------------------------------------------------------------------
2870017 typedef struct {
2870018     long int quot;
2870019     long int rem;
2870020 } ldiv_t;
2870021 //-----------------------------------------------------------------------
2870022 typedef void (*atexit_t) (void);        // Non standard. [1]
2870023 //
2870024 // [1] The type 'atexit_t' is a pointer to a function for the "at exit"
2870025 //     procedure, with no parameters and returning void. With the
2870026 //     declaration of type 'atexit_t', the function prototype of
2870027 //     'atexit()' is easier to declare and to understand. Original
2870028 //     declaration is:
2870029 //
2870030 //              int atexit (void (*function) (void));
2870031 //
2870032 //-----------------------------------------------------------------------
2870033 #define EXIT_FAILURE    1
2870034 #define EXIT_SUCCESS    0
2870035 #define RAND_MAX        INT_MAX
2870036 #define MB_CUR_MAX      ((size_t) MB_LEN_MAX)
2870037 //-----------------------------------------------------------------------
2870038 void              _Exit    (int status);
2870039 void              abort    (void);
2870040 int               abs      (int j);
2870041 int               atexit   (atexit_t function);
2870042 int               atoi     (const char *string);
2870043 long int          atol     (const char *string);
2870044 //void           *bsearch  (const void *key, const void *base,
2870045 //                          size_t nmemb, size_t size,
2870046 //                          int (*compar) (const void *, const void *));
2870047 #define           calloc(b, s) (malloc ((b) * (s)))
2870048 div_t             div      (int numer, int denom);
2870049 void              exit     (int status);
2870050 void              free     (void *ptr);
2870051 char             *getenv   (const char *name);
2870052 long int          labs     (long int j);
2870053 ldiv_t            ldiv     (long int numer, long int denom);
```



```
2870054 void             *malloc   (size_t size);
2870055 int               putenv   (const char *string);
2870056 void              qsort    (void *base, size_t nmemb, size_t size,
2870057                             int (*compare) (const void *,
2870058                                             const void *));
2870059 int               rand     (void);
2870060 void             *realloc  (void *ptr, size_t size);
2870061 int               setenv   (const char *name, const char *value,
2870062                             int overwrite);
2870063 void              srand    (unsigned int seed);
2870064 long int          strtol   (const char *restrict string,
2870065                             char **restrict endptr, int base);
2870066 unsigned long int strtoul  (const char * restrict string,
2870067                             char ** restrict endptr, int base);
2870068 //int             system   (const char *string);
2870069 int               unsetenv (const char *name);
2870070
2870071 #endif
```

lib/stdlib/_Exit.c

«

Si veda la sezione u0.2.

```
2880001 #include <stdlib.h>
2880002 #include <sys/os16.h>
2880003 //-----------------------------------------------------------------------
2880004 void
2880005 _Exit (int status)
2880006 {
2880007     sysmsg_exit_t msg;
2880008     //
2880009     // Only the low eight bit are returned.
2880010     //
2880011     msg.status = (status & 0xFF);
2880012     //
2880013     //
2880014     //
2880015     sys (SYS_EXIT, &msg, (sizeof msg));
2880016     //
2880017     // Should not return from system call, but if it does, loop
2880018     // forever:
2880019     //
2880020     while (1);
2880021 }
```

lib/stdlib/abort.c

«

Si veda la sezione u0.2.

```
2890001 #include <stdlib.h>
2890002 #include <sys/types.h>
2890003 #include <signal.h>
2890004 #include <unistd.h>
2890005 //-----------------------------------------------------------------------
2890006 void
2890007 abort (void)
2890008 {
2890009     pid_t        pid;
2890010     sighandler_t sig_previous;
2890011     //
2890012     // Set 'SIGABRT' to a default action.
2890013     //
2890014     sig_previous = signal (SIGABRT, SIG_DFL);
2890015     //
2890016     // If the previous action was something different than symbolic
2890017     // ones, configure again the previous action.
2890018     //
2890019     if (sig_previous != SIG_DFL &&
2890020         sig_previous != SIG_IGN &&
2890021         sig_previous != SIG_ERR)
2890022       {
2890023         signal (SIGABRT, sig_previous);
2890024       }
2890025     //
2890026     // Get current process ID and sent the signal.
2890027     //
2890028     pid = getpid ();
2890029     kill (pid, SIGABRT);
2890030     //
2890031     // Second chance
2890032     //
2890033     for (;;)
2890034       {
2890035         signal (SIGABRT, SIG_DFL);
2890036         pid = getpid ();
2890037         kill (pid, SIGABRT);
2890038       }
2890039 }
```

lib/stdlib/abs.c

«

Si veda la sezione u0.3.

```
2900001 #include <stdlib.h>
2900002 //-----------------------------------------------------------------------
2900003 int
2900004 abs (int j)
2900005 {
```

```
2900006     if (j < 0)
2900007       {
2900008         return -j;
2900009       }
2900010     else
2900011       {
2900012         return j;
2900013       }
2900014 }
```

lib/stdlib/alloc.c

«

Si veda la sezione u0.66.

```
2910001 #include <stdlib.h>
2910002 #include <string.h>
2910003 #include <errno.h>
2910004 #include <limits.h>
2910005 #include <stdio.h>
2910006 //---------------------------------------------------------------------
2910007 #define MEMORY_BLOCK_SIZE       1024
2910008 //---------------------------------------------------------------------
2910009 static char     _alloc_memory[LONG_BIT][MEMORY_BLOCK_SIZE];   // [1]
2910010 static size_t   _alloc_size[LONG_BIT];                        // [2]
2910011 static long int _alloc_map;                                   // [3]
2910012 //
2910013 // [1] Memory to be allocated.
2910014 // [2] Sizes allocated.
2910015 // [3] Memory block map. The memory map is made of a single integer and
2910016 //     the rightmost bit is the first memory block.
2910017 //---------------------------------------------------------------------
2910018 void *
2910019 malloc (size_t size)
2910020 {
2910021     size_t   size_free;   // Size free found that might be allocated.
2910022     int      m;           // Index inside '_alloc_memory[][]' table.
2910023     int      s;           // Start index for a free memory area.
2910024     long int mask;        // Mask to compare with '_alloc_map'.
2910025     long int alloc;       // New allocation map.
2910026     //
2910027     // Check for arguments.
2910028     //
2910029     if (size == 0)
2910030       {
2910031         return (NULL);
2910032       }
2910033     //
2910034     for (s = 0, m = 0; m < LONG_BIT; m++)
2910035       {
2910036         mask = 1;
2910037         mask <<= m;
2910038         //
2910039         if (_alloc_map & mask)
2910040           {
2910041             //
2910042             // The memory block is not free.
2910043             //
2910044             s         = m + 1;
2910045             size_free = 0;
2910046             alloc     = 0;
2910047           }
2910048         else
2910049           {
2910050             alloc     |= mask;
2910051             size_free += MEMORY_BLOCK_SIZE;
2910052           }
2910053         if (size_free >= size)
2910054           {
2910055             //
2910056             // Space found: update '_alloc_size[]' table, the map inside
2910057             // '_alloc_map' and return the memory address.
2910058             //
2910059             _alloc_size[s] = size_free;
2910060             _alloc_map    |= alloc;
2910061             return ((void *) &_alloc_memory[s][0]);
2910062           }
2910063       }
2910064     //
2910065     // No space left.
2910066     //
2910067     errset (ENOMEM);                    // Not enough space.
2910068     //
2910069     return (NULL);
2910070 }
2910071 //---------------------------------------------------------------------
2910072 void
2910073 free (void *address)
2910074 {
2910075     size_t   size_free;  // Size to make free.
2910076     int      m;          // Index inside '_alloc_memory[][]' table.
2910077     int      s;          // Start index.
2910078     long int mask;       // Mask to compare with '_alloc_map'.
2910079     long int alloc;      // New allocation map.
2910080     //
2910081     // Check argument.
2910082     //
2910083     if (address == NULL)
2910084       {
2910085         return;
2910086       }
2910087     //
```



```
2910088     // Find the original allocated address inside '_alloc_memory[][]'
2910089     // table.
2910090     //
2910091     for (m = 0; m < LONG_BIT; m++)
2910092       {
2910093         if (address == (void *) &_alloc_memory[m][0])
2910094           {
2910095             //
2910096             // This is the right memory block.
2910097             //
2910098             if (_alloc_size[m] == 0)
2910099               {
2910100                 //
2910101                 // The block found is not allocated.
2910102                 //
2910103                 return;
2910104               }
2910105             else
2910106               {
2910107                 //
2910108                 // Build the map of the memory to set free.
2910109                 //
2910110                 size_free = _alloc_size[m];
2910111                 for (alloc = 0, s = m;
2910112                      size_free > 0 && s < LONG_BIT;
2910113                      size_free -= MEMORY_BLOCK_SIZE, s++)
2910114                   {
2910115                     mask   = 1;
2910116                     mask <<= s;
2910117                     alloc |= mask;
2910118                   }
2910119                 //
2910120                 // Compare the map of memory to be freed with the
2910121                 // reality allocated one, then free the memory.
2910122                 //
2910123                 if ((_alloc_map & alloc) == alloc)
2910124                   {
2910125                     _alloc_map    &= ~alloc;
2910126                     _alloc_size[m] = 0;
2910127                     return;
2910128                   }
2910129                 //
2910130                 // The real map does not report the same amount of
2910131                 // allocated memory, so nothing is freed.
2910132                 //
2910133                 return;
2910134               }
2910135           }
2910136       }
2910137     //
2910138     // Address not allocated.
2910139     //
2910140     return;
2910141 }
2910142 //---------------------------------------------------------------------
2910143 void *
2910144 realloc (void *address, size_t size)
2910145 {
2910146     char  *address_new;
2910147     char  *address_old = (char *) address;
2910148     size_t size_old    = 0;
2910149     size_t size_new    = size;
2910150     int    m;                   // Index inside the memory table;
2910151     //
2910152     // Check arguments.
2910153     //
2910154     if (size == 0)              return (NULL);
2910155     if (address == NULL)        return (malloc (size));
2910156     //
2910157     // Locate original allocation.
2910158     //
2910159     for (m = 0; m < LONG_BIT; m++)
2910160       {
2910161         if (address_old == (char *) &_alloc_memory[m][0])
2910162           {
2910163             size_old = _alloc_size[m];
2910164             break;
2910165           }
2910166       }
2910167     //
2910168     // Check if a valid size was found.
2910169     //
2910170     if (size_old == 0)
2910171       {
2910172         //
2910173         // Address not found or size not valid.
2910174         //
2910175         return (NULL);
2910176       }
2910177     //
2910178     // Allocate the new memory.
2910179     //
2910180     address_new = malloc (size);
2910181     //
2910182     // Check allocation. If there is an error, the variable 'errno'
2910183     // is already updated by 'malloc()'.
2910184     //
2910185     if (address_new == NULL)
2910186       {
2910187         return (NULL);
2910188       }
```

```
2910189     //
2910190     // Copy old memory.
2910191     //
2910192     for (; size_old > 0 && size_new > 0;
2910193          size_old--, size_new--, address_new++, address_old++)
2910194       {
2910195        *address_new = *address_old;
2910196       }
2910197     //
2910198     // Free old memory.
2910199     //
2910200     free (address);
2910201     //
2910202     // Return the new address.
2910203     //
2910204     return (address_new);
2910205 }
2910206 //---------------------------------------------------------------------
```

lib/stdlib/atexit.c

«

Si veda la sezione u0.4.

```
2920001 #include <stdlib.h>
2920002 #include <stdio.h>
2920003 //---------------------------------------------------------------------
2920004 atexit_t _atexit_table[ATEXIT_MAX];
2920005 //---------------------------------------------------------------------
2920006 void
2920007 _atexit_setup (void)
2920008 {
2920009     int a;
2920010     //
2920011     for (a = 0; a < ATEXIT_MAX; a++)
2920012       {
2920013         _atexit_table[a] = NULL;
2920014       }
2920015 }
2920016 //---------------------------------------------------------------------
2920017 int
2920018 atexit (atexit_t function)
2920019 {
2920020     int a;
2920021     //
2920022     if (function == NULL)
2920023       {
2920024         return (-1);
2920025       }
2920026     //
2920027     for (a = 0; a < ATEXIT_MAX; a++)
2920028       {
2920029         if (_atexit_table[a] == NULL)
2920030           {
2920031             _atexit_table[a] = function;
2920032             return (0);
2920033           }
2920034       }
2920035     //
2920036     return (-1);
2920037 }
```

lib/stdlib/atoi.c

«

Si veda la sezione u0.5.

```
2930001 #include <stdlib.h>
2930002 #include <ctype.h>
2930003 //---------------------------------------------------------------------
2930004 int
2930005 atoi (const char *string)
2930006 {
2930007     int i;
2930008     int sign = +1;
2930009     int number;
2930010     //
2930011     for (i = 0; isspace (string[i]); i++)
2930012       {
2930013         ;
2930014       }
2930015     //
2930016     if (string[i] == '+')
2930017       {
2930018         sign = +1;
2930019         i++;
2930020       }
2930021     else if (string[i] == '-')
2930022       {
2930023         sign = -1;
2930024         i++;
2930025       }
2930026     //
2930027     for (number = 0; isdigit (string[i]); i++)
2930028       {
2930029         number *= 10;
2930030         number += (string[i] - '0');
2930031       }
2930032     //
2930033     number *= sign;
2930034     //
2930035     return number;
```



```
2930036 }
```

lib/stdlib/atol.c

«

Si veda la sezione u0.5.

```
2940001 #include <stdlib.h>
2940002 #include <ctype.h>
2940003 //---------------------------------------------------------------------
2940004 long int
2940005 atol (const char *string)
2940006 {
2940007     int      i;
2940008     int      sign = +1;
2940009     long int number;
2940010     //
2940011     for (i = 0; isspace (string[i]); i++)
2940012       {
2940013         ;
2940014       }
2940015     //
2940016     if (string[i] == '+')
2940017       {
2940018         sign = +1;
2940019         i++;
2940020       }
2940021     else if (string[i] == '-')
2940022       {
2940023         sign = -1;
2940024         i++;
2940025       }
2940026     //
2940027     for (number = 0; isdigit (string[i]); i++)
2940028       {
2940029         number *= 10;
2940030         number += (string[i] - '0');
2940031       }
2940032     //
2940033     number *= sign;
2940034     //
2940035     return number;
2940036 }
```

lib/stdlib/div.c

«

Si veda la sezione u0.15.

```
2950001 #include <stdlib.h>
2950002 //---------------------------------------------------------------------
2950003 div_t
2950004 div (int numer, int denom)
2950005 {
2950006     div_t d;
2950007     d.quot = numer / denom;
2950008     d.rem = numer % denom;
2950009     return d;
2950010 }
```

lib/stdlib/environment.c

«

Si veda la sezione u0.1.

```
2960001 #include <stdlib.h>
2960002 #include <string.h>
2960003 //---------------------------------------------------------------------
2960004 // This file contains a non standard definition, related to the
2960005 // environment handling.
2960006 //
2960007 // The file 'crt0.s', before calling the main function, calls the
2960008 // function '_environment_setup(), that is responsible for initializing
2960009 // the array '_environment_table[][]' and for copying the content
2960010 // of the environment, as it comes from the 'exec()' system call.
2960011 //
2960012 // The pointers to the environment strings organised inside the
2960013 // array '_environment_table[][]', are also copied inside the
2960014 // array of pointers '_environment[]'.
2960015 //
2960016 // After all that is done, inside 'crt0.s', the pointer to
2960017 // '_environment[]' is copied to the traditional variable 'environ'
2960018 // and also to the previous value of the pointer variable 'envp'.
2960019 //
2960020 // This way, applications will get the environment, but organised
2960021 // inside the table '_environment_table[][]'. So, functions like
2960022 // 'getenv()' and 'setenv()' do know where to look for.
2960023 //
2960024 // It is useful to notice that there is no prototype and no extern
2960025 // declaration inside the file <stdlib.h>, about this function
2960026 // and these arrays, because applications do not have to know about
2960027 // it.
2960028 //
2960029 // Please notice that 'environ' could be just the same as
2960030 // '_environment' here, but the common use puts 'environ' inside
2960031 // <unistd.h>, although for this implementation it should be better
2960032 // placed inside <stdlib.h>.
2960033 //
2960034 //---------------------------------------------------------------------
2960035 char  _environment_table[ARG_MAX/32][ARG_MAX/16];
2960036 char *_environment[ARG_MAX/32+1];
```

```
2960037 //------------------------------------------------------------------------
2960038 void
2960039 _environment_setup (char *envp[])
2960040 {
2960041     int e;
2960042     int s;
2960043     //
2960044     // Reset the '_environment_table[][]' array.
2960045     //
2960046     for (e = 0; e < ARG_MAX/32; e++)
2960047       {
2960048         for (s = 0; s < ARG_MAX/16; s++)
2960049           {
2960050             _environment_table[e][s] = 0;
2960051           }
2960052       }
2960053     //
2960054     // Set the '_environment[]' pointers. The final extra element must
2960055     // be a NULL pointer.
2960056     //
2960057     for (e = 0; e < ARG_MAX/32 ; e++)
2960058       {
2960059         _environment[e] = _environment_table[e];
2960060       }
2960061     _environment[ARG_MAX/32] = NULL;
2960062     //
2960063     // Copy the environment inside the array, but only if 'envp' is
2960064     // not NULL.
2960065     //
2960066     if (envp != NULL)
2960067       {
2960068         for (e = 0; envp[e] != NULL && e < ARG_MAX/32; e++)
2960069           {
2960070             strncpy (_environment_table[e], envp[e], (ARG_MAX/16)-1);
2960071           }
2960072       }
2960073 }
```

lib/stdlib/exit.c

«

Si veda la sezione u0.4.

```
2970001 #include <stdlib.h>
2970002 #include <stdio.h>
2970003 //------------------------------------------------------------------------
2970004 extern atexit_t _atexit_table[];
2970005 //------------------------------------------------------------------------
2970006 void
2970007 exit (int status)
2970008 {
2970009     int a;
2970010     //
2970011     // The "at exit" functions must be called in reverse order.
2970012     //
2970013     for (a = (ATEXIT_MAX - 1); a >= 0; a--)
2970014       {
2970015         if (_atexit_table[a] != NULL)
2970016           {
2970017             (*_atexit_table[a]) ();
2970018           }
2970019       }
2970020     //
2970021     // Now: really exit.
2970022     //
2970023     _Exit (status);
2970024     //
2970025     // Should not return from system call, but if it does, loop
2970026     // forever:
2970027     //
2970028     while (1);
2970029 }
```

lib/stdlib/getenv.c

«

Si veda la sezione u0.51.

```
2980001 #include <stdlib.h>
2980002 #include <string.h>
2980003 //------------------------------------------------------------------------
2980004 extern char *_environment[];
2980005 //------------------------------------------------------------------------
2980006 char *
2980007 getenv (const char *name)
2980008 {
2980009     int   e;        // First index: environment table items.
2980010     int   f;        // Second index: environment string scan.
2980011     char *value;    // Pointer to the environment value found.
2980012     //
2980013     // Check if the input is valid. No error is reported.
2980014     //
2980015     if (name == NULL || strlen (name) == 0)
2980016       {
2980017         return (NULL);
2980018       }
2980019     //
2980020     // Scan the environment table items, with index 'e'. The pointer
2980021     // 'value' is initialized to NULL. If the pointer 'value' gets a
2980022     // valid  pointer, the environment variable was found and a
2980023     // pointer to the beginning of its value is available.
2980024     //
```



```
2980025     for (value = NULL, e = 0; e < ARG_MAX/32; e++)
2980026       {
2980027         //
2980028         // Scan the string of the environment item, with index 'f'.
2980029         // The scan continue until 'name[f]' and '_environment[e][f]'
2980030         // are equal.
2980031         //
2980032         for (f = 0;
2980033              f < ARG_MAX/16-1 && name[f] == _environment[e][f];
2980034              f++)
2980035           {
2980036             ; // Just scan.
2980037           }
2980038         //
2980039         // At this point, 'name[f]' and '_environment[e][f]' are
2980040         // different: if 'name[f]' is zero the name string is
2980041         // terminated; if '_environment[e][f]' is also equal to '=',
2980042         // the environment item is corresponding to the requested name.
2980043         //
2980044         if (name[f] == 0 && _environment[e][f] == '=')
2980045           {
2980046             //
2980047             // The pointer to the beginning of the environment value is
2980048             // calculated, and the external loop exit.
2980049             //
2980050             value = &_environment[e][f+1];
2980051             break;
2980052           }
2980053       }
2980054     //
2980055     // The 'value' is returned: if it is still NULL, then, no
2980056     // environment variable with the requested name was found.
2980057     //
2980058     return (value);
2980059 }
```

lib/stdlib/labs.c

«

Si veda la sezione u0.3.

```
2990001 #include <stdlib.h>
2990002 //------------------------------------------------------------------------
2990003 long int
2990004 labs (long int j)
2990005 {
2990006     if (j < 0)
2990007       {
2990008         return -j;
2990009       }
2990010     else
2990011       {
2990012         return j;
2990013       }
2990014 }
```

lib/stdlib/ldiv.c

«

Si veda la sezione u0.15.

```
3000001 #include <stdlib.h>
3000002 //------------------------------------------------------------------------
3000003 ldiv_t
3000004 ldiv (long int numer, long int denom)
3000005 {
3000006     ldiv_t d;
3000007     d.quot = numer / denom;
3000008     d.rem = numer % denom;
3000009     return d;
3000010 }
```

lib/stdlib/putenv.c

«

Si veda la sezione u0.82.

```
3010001 #include <stdlib.h>
3010002 #include <string.h>
3010003 #include <errno.h>
3010004 //------------------------------------------------------------------------
3010005 extern char *_environment[];
3010006 //------------------------------------------------------------------------
3010007 int
3010008 putenv (const char *string)
3010009 {
3010010     int   e;        // First index: environment table items.
3010011     int   f;        // Second index: environment string scan.
3010012     //
3010013     // Check if the input is empty. No error is reported.
3010014     //
3010015     if (string == NULL || strlen (string) == 0)
3010016       {
3010017         return (0);
3010018       }
3010019     //
3010020     // Check if the input is valid: there must be a '=' sign.
3010021     // Error here is reported.
3010022     //
3010023     if (strchr (string, '=') == NULL)
3010024       {
```

```
3010025         errset(EINVAL);                 // Invalid argument.
3010026         return (-1);
3010027       }
3010028     //
3010029     // Scan the environment table items, with index 'e'. The intent is
3010030     // to find a previous environment variable with the same name.
3010031     //
3010032     for (e = 0; e < ARG_MAX/32; e++)
3010033       {
3010034         //
3010035         // Scan the string of the environment item, with index 'f'.
3010036         // The scan continue until 'string[f]' and '_environment[e][f]'
3010037         // are equal.
3010038         //
3010039         for (f = 0;
3010040              f < ARG_MAX/16-1 && string[f] == _environment[e][f];
3010041              f++)
3010042           {
3010043             ; // Just scan.
3010044           }
3010045         //
3010046         // At this point, 'string[f-1]' and '_environment[e][f-1]'
3010047         // should contain '='. If it is so, the environment is replaced.
3010048         //
3010049         if (string[f-1] == '=' && _environment[e][f-1] == '=')
3010050           {
3010051             //
3010052             // The environment item was found: now replace the pointer.
3010053             //
3010054             _environment[e] = string;
3010055             //
3010056             // Return.
3010057             //
3010058             return (0);
3010059           }
3010060       }
3010061     //
3010062     // The item was not found. Scan again for a free slot.
3010063     //
3010064     for (e = 0; e < ARG_MAX/32; e++)
3010065       {
3010066         if (_environment[e] == NULL || _environment[e][0] == 0)
3010067           {
3010068             //
3010069             // An empty item was found and the pointer will be
3010070             // replaced.
3010071             //
3010072             _environment[e] = string;
3010073             //
3010074             // Return.
3010075             //
3010076             return (0);
3010077           }
3010078       }
3010079     //
3010080     // Sorry: the empty slot was not found!
3010081     //
3010082     errset (ENOMEM);            // Not enough space.
3010083     return (-1);
3010084 }
```

lib/stdlib/qsort.c

« Si veda la sezione u0.84.

```
3020001 #include <stdlib.h>
3020002 #include <string.h>
3020003 #include <errno.h>
3020004 //---------------------------------------------------------------------
3020005 static int  part (char *array, size_t size, int a, int z,
3020006                   int (*compare)(const void *, const void *));
3020007 static void sort (char *array, size_t size, int a, int z,
3020008                   int (*compare)(const void *, const void *));
3020009 //---------------------------------------------------------------------
3020010 void
3020011 qsort (void *base, size_t nmemb, size_t size,
3020012        int (*compare)(const void *, const void *))
3020013 {
3020014     if (size <= 1)
3020015       {
3020016         //
3020017         // There is nothing to sort!
3020018         //
3020019         return;
3020020       }
3020021     else
3020022       {
3020023         sort ((char *) base, size, 0, (int) (nmemb - 1), compare);
3020024       }
3020025 }
3020026 //---------------------------------------------------------------------
3020027 static void
3020028 sort (char *array, size_t size, int a, int z,
3020029       int (*compare)(const void *, const void *))
3020030 {
3020031     int loc;
3020032     //
3020033     if (z > a)
3020034       {
3020035         loc = part (array, size, a, z, compare);
3020036         if (loc >= 0)
```



```
3020037           {
3020038             sort (array, size, a, loc-1, compare);
3020039             sort (array, size, loc+1, z, compare);
3020040           }
3020041       }
3020042 }
3020043
3020044 //---------------------------------------------------------------------
3020045 static int
3020046 part (char *array, size_t size, int a, int z,
3020047       int (*compare)(const void *, const void *))
3020048 {
3020049     int   i;
3020050     int   loc;
3020051     char *swap;
3020052     //
3020053     if (z <= a)
3020054       {
3020055         errset (EUNKNOWN);       // Should never happen.
3020056         return (-1);
3020057       }
3020058     //
3020059     // Index 'i' after the first element; index 'loc' at the last
3020060     // position.
3020061     //
3020062     i   = a + 1;
3020063     loc = z;
3020064     //
3020065     // Prepare space in memory for element swap.
3020066     //
3020067     swap = malloc (size);
3020068     if (swap == NULL)
3020069       {
3020070         errset (ENOMEM);
3020071         return (-1);
3020072       }
3020073     //
3020074     // Loop as long as index 'loc' is higher than index 'i'.
3020075     // When index 'loc' is less or equal to index 'i',
3020076     // then, index 'loc' is the right position for the
3020077     // first element of the current piece of array.
3020078     //
3020079     for (;;)
3020080       {
3020081         //
3020082         // Index 'i' goes up...
3020083         //
3020084         for (;i < loc; i++)
3020085           {
3020086             if  (compare (&array[i*size], &array[a*size]) > 0)
3020087               {
3020088                 break;
3020089               }
3020090           }
3020091         //
3020092         // Index 'loc' gose down...
3020093         //
3020094         for (;; loc--)
3020095           {
3020096             if  (compare (&array[loc*size], &array[a*size]) <= 0)
3020097               {
3020098                 break;
3020099               }
3020100           }
3020101         //
3020102         // Swap elements related to index 'i' and 'loc'.
3020103         //
3020104         if (loc <= i)
3020105           {
3020106             //
3020107             // The array is completely scanned.
3020108             //
3020109             break;
3020110           }
3020111         else
3020112           {
3020113             memcpy (swap, &array[loc*size], size);
3020114             memcpy (&array[loc*size], &array[i*size], size);
3020115             memcpy (&array[i*size], swap, size);
3020116           }
3020117       }
3020118     //
3020119     // Swap the first element with the one related to the
3020120     // index 'loc'.
3020121     //
3020122     memcpy (swap, &array[loc*size], size);
3020123     memcpy (&array[loc*size], &array[a*size], size);
3020124     memcpy (&array[a*size], swap, size);
3020125     //
3020126     // Free the swap memory.
3020127     //
3020128     free (swap);
3020129     //
3020130     // Return the index 'loc'.
3020131     //
3020132     return (loc);
3020133 }
3020134
```

lib/stdlib/rand.c

«

Si veda la sezione u0.85.

```
3030001 #include <stdlib.h>
3030002 //---------------------------------------------------------------------
3030003 static unsigned int _srand = 1; // The '_srand' rank must be at least
3030004                                  // 'unsigned int' and must be able to
3030005                                  // represent the value 'RAND_MAX'.
3030006 //---------------------------------------------------------------------
3030007 int
3030008 rand (void)
3030009 {
3030010     _srand = _srand * 12345 + 123;
3030011     return _srand % ((unsigned int) RAND_MAX + 1);
3030012 }
3030013 //---------------------------------------------------------------------
3030014 void
3030015 srand (unsigned int seed)
3030016 {
3030017     _srand = seed;
3030018 }
```

lib/stdlib/setenv.c

«

Si veda la sezione u0.94.

```
3040001 #include <stdlib.h>
3040002 #include <string.h>
3040003 #include <errno.h>
3040004 //---------------------------------------------------------------------
3040005 extern char *_environment[];
3040006 extern char *_environment_table[];
3040007 //---------------------------------------------------------------------
3040008 int
3040009 setenv (const char *name, const char *value, int overwrite)
3040010 {
3040011     int   e;        // First index: environment table items.
3040012     int   f;        // Second index: environment string scan.
3040013     //
3040014     // Check if the input is empty. No error is reported.
3040015     //
3040016     if (name == NULL || strlen (name) == 0)
3040017       {
3040018         return (0);
3040019       }
3040020     //
3040021     // Check if the input is valid: error here is reported.
3040022     //
3040023     if (strchr (name, '=') != NULL)
3040024       {
3040025         errset(EINVAL);                 // Invalid argument.
3040026         return (-1);
3040027       }
3040028     //
3040029     // Check if the input is too big.
3040030     //
3040031     if ((strlen (name) + strlen (value) + 2) > ARG_MAX/16)
3040032       {
3040033         //
3040034         // The environment to be saved is bigger than the
3040035         // available string size, inside '_environment_table[]'.
3040036         //
3040037         errset (ENOMEM);                // Not enough space.
3040038         return (-1);
3040039       }
3040040     //
3040041     // Scan the environment table items, with index 'e'. The intent is
3040042     // to find a previous environment variable with the same name.
3040043     //
3040044     for (e = 0; e < ARG_MAX/32; e++)
3040045       {
3040046         //
3040047         // Scan the string of the environment item, with index 'f'.
3040048         // The scan continue until 'name[f]' and '_environment[e][f]'
3040049         // are equal.
3040050         //
3040051         for (f = 0;
3040052              f < ARG_MAX/16-1 && name[f] == _environment[e][f];
3040053              f++)
3040054           {
3040055             ; // Just scan.
3040056           }
3040057         //
3040058         // At this point, 'name[f]' and '_environment[e][f]' are
3040059         // different: if 'name[f]' is zero the name string is
3040060         // terminated; if '_environment[e][f]' is also equal to '=',
3040061         // the environment item is corresponding to the requested name.
3040062         //
3040063         if (name[f] == 0 && _environment[e][f] == '=')
3040064           {
3040065             //
3040066             // The environment item was found; if it can be overwritten,
3040067             // the write is done.
3040068             //
3040069             if (overwrite)
3040070               {
3040071                 //
3040072                 // To be able to handle both 'setenv()' and 'putenv()',
3040073                 // before removing the item, it is fixed the pointer to
3040074                 // the global environment table.
```



```
3040075                 //
3040076                 _environment[e] = _environment_table[e];
3040077                 //
3040078                 // Now copy the new environment. The string size was
3040079                 // already checked.
3040080                 //
3040081                 strcpy (_environment[e], name);
3040082                 strcat (_environment[e], "=");
3040083                 strcat (_environment[e], value);
3040084                 //
3040085                 // Return.
3040086                 //
3040087                 return (0);
3040088               }
3040089             //
3040090             // Cannot overwrite!
3040091             //
3040092             errset (EUNKNOWN);
3040093             return (-1);
3040094           }
3040095       }
3040096     //
3040097     // The item was not found. Scan again for a free slot.
3040098     //
3040099     for (e = 0; e < ARG_MAX/32; e++)
3040100       {
3040101         if (_environment[e] == NULL || _environment[e][0] == 0)
3040102           {
3040103             //
3040104             // An empty item was found. To be able to handle both
3040105             // 'setenv()' and 'putenv()', it is fixed the pointer to
3040106             // the global environment table.
3040107             //
3040108             _environment[e] = _environment_table[e];
3040109             //
3040110             // Now copy the new environment. The string size was
3040111             // already checked.
3040112             //
3040113             strcpy (_environment[e], name);
3040114             strcat (_environment[e], "=");
3040115             strcat (_environment[e], value);
3040116             //
3040117             // Return.
3040118             //
3040119             return (0);
3040120           }
3040121       }
3040122     //
3040123     // Sorry: the empty slot was not found!
3040124     //
3040125     errset (ENOMEM);            // Not enough space.
3040126     return (-1);
3040127 }
```

lib/stdlib/strtol.c

«

Si veda la sezione u0.121.

```
3050001 #include <stdlib.h>
3050002 #include <ctype.h>
3050003 #include <errno.h>
3050004 #include <limits.h>
3050005 #include <stdbool.h>
3050006 //---------------------------------------------------------------------
3050007 #define isoctal(C)  ((int) (C >= '0' && C <= '7'))
3050008 //---------------------------------------------------------------------
3050009 long int
3050010 strtol (const char *restrict string, char **restrict endptr, int base)
3050011 {
3050012     int      i;
3050013     int      sign = +1;
3050014     long int number;
3050015     long int previous;
3050016     int      digit;
3050017     //
3050018     bool     flag_prefix_oct = 0;
3050019     bool     flag_prefix_exa = 0;
3050020     bool     flag_prefix_dec = 0;
3050021     //
3050022     // Check base and string.
3050023     //
3050024     if (base < 0
3050025      || base > 36
3050026      || base == 1               // With base 1 cannot do anything.
3050027      || string == NULL
3050028      || string[0] == 0)
3050029       {
3050030         if (endptr != NULL) *endptr = string;
3050031         errset (EINVAL);               // Invalid argument.
3050032         return ((long int) 0);
3050033       }
3050034     //
3050035     // Eat initial spaces.
3050036     //
3050037     for (i = 0; isspace (string[i]); i++)
3050038       {
3050039         ;
3050040       }
3050041     //
3050042     // Check sign.
3050043     //
```

```
3050044     if (string[i] == '+')
3050045       {
3050046         sign = +1;
3050047         i++;
3050048       }
3050049     else if (string[i] == '-')
3050050       {
3050051         sign = -1;
3050052         i++;
3050053       }
3050054     //
3050055     // Check for prefix.
3050056     //
3050057     if (string[i] == '0')
3050058       {
3050059         if (string[i+1] == 'x' || string[i+1] == 'X')
3050060           {
3050061             flag_prefix_exa = 1;
3050062           }
3050063         else if (isoctal (string[i+1]))
3050064           {
3050065             flag_prefix_oct = 1;
3050066           }
3050067         else
3050068           {
3050069             flag_prefix_dec = 1;
3050070           }
3050071       }
3050072     else if (isdigit (string[i]))
3050073       {
3050074         flag_prefix_dec = 1;
3050075       }
3050076     //
3050077     // Check compatibility with requested base.
3050078     //
3050079     if (flag_prefix_exa)
3050080       {
3050081         //
3050082         // At the moment, there is a zero and a 'x'. Might be
3050083         // exadecimal, or might be a number base 33 or more.
3050084         //
3050085         if (base == 0)
3050086           {
3050087             base = 16;
3050088           }
3050089         else if (base == 16)
3050090           {
3050091             ;   // Ok.
3050092           }
3050093         else if (base >= 33)
3050094           {
3050095             ;   // Ok.
3050096           }
3050097         else
3050098           {
3050099             //
3050100             // Incompatible sequence: only the initial zero is reported.
3050101             //
3050102             if (endptr != NULL) *endptr = &string[i+1];
3050103             return ((long int) 0);
3050104           }
3050105         //
3050106         // Move on, after the '0x' prefix.
3050107         //
3050108         i += 2;
3050109       }
3050110     //
3050111     if (flag_prefix_oct)
3050112       {
3050113         //
3050114         // There is a zero and a digit.
3050115         //
3050116         if (base == 0)
3050117           {
3050118             base = 8;
3050119           }
3050120         //
3050121         // Move on, after the '0' prefix.
3050122         //
3050123         i += 1;
3050124       }
3050125     //
3050126     if (flag_prefix_dec)
3050127       {
3050128         if (base == 0)
3050129           {
3050130             base = 10;
3050131           }
3050132       }
3050133     //
3050134     // Scan the string.
3050135     //
3050136     for (number = 0; string[i] != 0; i++)
3050137       {
3050138         if      (string[i] >= '0' && string[i] <= '9')
3050139           {
3050140             digit = string[i] - '0';
3050141           }
3050142         else if (string[i] >= 'A' && string[i] <= 'Z')
3050143           {
3050144             digit = string[i] - 'A' + 10;
```



```
3050145           }
3050146         else if (string[i] >= 'a' && string[i] <= 'z')
3050147           {
3050148             digit = string[i] - 'a' + 10;
3050149           }
3050150         else
3050151           {
3050152             //
3050153             // This is an out of range digit.
3050154             //
3050155             digit = 999;
3050156           }
3050157         //
3050158         // Give a sign to the digit.
3050159         //
3050160         digit *= sign;
3050161         //
3050162         // Compare with the base.
3050163         //
3050164         if (base > (digit * sign))
3050165           {
3050166             //
3050167             // Check if the current digit can be safely computed.
3050168             //
3050169             previous = number;
3050170             number  *= base;
3050171             number  += digit;
3050172             if (number / base != previous)
3050173               {
3050174                 //
3050175                 // Out of range.
3050176                 //
3050177                 if (endptr != NULL) *endptr = &string[i+1];
3050178                 errset (ERANGE);        // Result too large.
3050179                 if (sign > 0)
3050180                   {
3050181                     return (LONG_MAX);
3050182                   }
3050183                 else
3050184                   {
3050185                     return (LONG_MIN);
3050186                   }
3050187               }
3050188           }
3050189         else
3050190           {
3050191             if (endptr != NULL) *endptr = &string[i];
3050192             return (number);
3050193           }
3050194       }
3050195     //
3050196     // The string is finished.
3050197     //
3050198     if (endptr != NULL) *endptr = &string[i];
3050199     //
3050200     return (number);
3050201 }
```

lib/stdlib/strtoul.c

«

Si veda la sezione u0.121.

```
3060001 #include <stdlib.h>
3060002 #include <ctype.h>
3060003 #include <errno.h>
3060004 #include <limits.h>
3060005 //------------------------------------------------------------------------
3060006 // A really poor implementation. ,-(
3060007 //
3060008 unsigned long int
3060009 strtoul (const char *restrict string, char **restrict endptr, int base)
3060010 {
3060011     return ((unsigned long int) strtol (string, endptr, base));
3060012 }
```

lib/stdlib/unsetenv.c

«

Si veda la sezione u0.94.

```
3070001 #include <stdlib.h>
3070002 #include <string.h>
3070003 #include <errno.h>
3070004 //------------------------------------------------------------------------
3070005 extern char *_environment[];
3070006 extern char *_environment_table[];
3070007 //------------------------------------------------------------------------
3070008 int
3070009 unsetenv (const char *name)
3070010 {
3070011     int   e;        // First index: environment table items.
3070012     int   f;        // Second index: environment string scan.
3070013     //
3070014     // Check if the input is empty. No error is reported.
3070015     //
3070016     if (name == NULL || strlen (name) == 0)
3070017       {
3070018         return (0);
3070019       }
3070020     //
3070021     // Check if the input is valid: error here is reported.
```

```
3070022     //
3070023     if (strchr (name, '=') != NULL)
3070024       {
3070025         errset(EINVAL);                 // Invalid argument.
3070026         return (-1);
3070027       }
3070028     //
3070029     // Scan the environment table items, with index 'e'.
3070030     //
3070031     for (e = 0; e < ARG_MAX/32; e++)
3070032       {
3070033         //
3070034         // Scan the string of the environment item, with index 'f'.
3070035         // The scan continue until 'name[f]' and '_environment[e][f]'
3070036         // are equal.
3070037         //
3070038         for (f = 0;
3070039              f < ARG_MAX/16-1 && name[f] == _environment[e][f];
3070040              f++)
3070041           {
3070042             ; // Just scan.
3070043           }
3070044         //
3070045         // At this point, 'name[f]' and '_environment[e][f]' are
3070046         // different: if 'name[f]' is zero the name string is
3070047         // terminated; if '_environment[e][f]' is also equal to '=',
3070048         // the environment item is corresponding to the requested name.
3070049         //
3070050         if (name[f] == 0 && _environment[e][f] == '=')
3070051           {
3070052             //
3070053             // The environment item was found and it have to be removed.
3070054             // To be able to handle both 'setenv()' and 'putenv()',
3070055             // before removing the item, it is fixed the pointer to
3070056             // the global environment table.
3070057             //
3070058             _environment[e] = _environment_table[e];
3070059             //
3070060             // Now remove the environment item.
3070061             //
3070062             _environment[e][0] = 0;
3070063             break;
3070064           }
3070065       }
3070066     //
3070067     // Work done fine.
3070068     //
3070069     return (0);
3070070 }
```

os16: «lib/string.h»

«

Si veda la sezione u0.2.

```
3080001 #ifndef _STRING_H
3080002 #define _STRING_H       1
3080003
3080004 #include <const.h>
3080005 #include <restrict.h>
3080006 #include <const.h>
3080007 #include <size_t.h>
3080008 #include <NULL.h>
3080009 //---------------------------------------------------------------------
3080010 void  *memccpy (void *restrict dst, const void *restrict org, int c,
3080011                 size_t n);
3080012 void  *memchr  (const void *memory, int c, size_t n);
3080013 int    memcmp  (const void *memory1, const void *memory2, size_t n);
3080014 void  *memcpy  (void *restrict dst, const void *restrict org, size_t n);
3080015 void  *memmove (void *dst, const void *org, size_t n);
3080016 void  *memset  (void *memory, int c, size_t n);
3080017 char  *strcat  (char *restrict dst, const char *restrict org);
3080018 char  *strchr  (const char *string, int c);
3080019 int    strcmp  (const char *string1, const char *string2);
3080020 int    strcoll (const char *string1, const char *string2);
3080021 char  *strcpy  (char *restrict dst, const char *restrict org);
3080022 size_t strcspn (const char *string, const char *reject);
3080023 char  *strdup  (const char *string);
3080024 char  *strerror (int errnum);
3080025 size_t strlen  (const char *string);
3080026 char  *strncat (char *restrict dst, const char *restrict org, size_t n);
3080027 int    strncmp (const char *string1, const char *string2, size_t n);
3080028 char  *strncpy (char *restrict dst, const char *restrict org, size_t n);
3080029 char  *strpbrk (const char *string, const char *accept);
3080030 char  *strrchr (const char *string, int c);
3080031 size_t strspn  (const char *string, const char *accept);
3080032 char  *strstr  (const char *string, const char *substring);
3080033 char  *strtok  (char *restrict string, const char *restrict delim);
3080034 size_t strxfrm (char *restrict dst, const char *restrict org, size_t n);
3080035 //---------------------------------------------------------------------
3080036
3080037
3080038
3080039 #endif
```


lib/string/memccpy.c

«

Si veda la sezione u0.67.

```
3090001 #include <string.h>
3090002 //---------------------------------------------------------------------
3090003 void *
3090004 memccpy (void *restrict dst, const void *restrict org, int c, size_t n)
3090005 {
3090006     char *d = (char *) dst;
3090007     char *o = (char *) org;
3090008     size_t i;
3090009     for (i = 0; n > 0 && i < n; i++)
3090010       {
3090011         d[i] = o[i];
3090012         if (d[i] == (char) c)
3090013           {
3090014             return ((void *) &d[i+1]);
3090015           }
3090016       }
3090017     return (NULL);
3090018 }
```

lib/string/memchr.c

«

Si veda la sezione u0.68.

```
3100001 #include <string.h>
3100002 //---------------------------------------------------------------------
3100003 void *
3100004 memchr (const void *memory, int c, size_t n)
3100005 {
3100006     char *m = (char *) memory;
3100007     size_t i;
3100008     for (i = 0; n > 0 && i < n; i++)
3100009       {
3100010         if (m[i] == (char) c)
3100011           {
3100012             return (void *) (m + i);
3100013           }
3100014       }
3100015     return NULL;
3100016 }
```

lib/string/memcmp.c

«

Si veda la sezione u0.69.

```
3110001 #include <string.h>
3110002 //---------------------------------------------------------------------
3110003 int
3110004 memcmp (const void *memory1, const void *memory2, size_t n)
3110005 {
3110006     char *a = (char *) memory1;
3110007     char *b = (char *) memory2;
3110008     size_t i;
3110009     for (i = 0; n > 0 && i < n; i++)
3110010       {
3110011         if (a[i] > b[i])
3110012           {
3110013             return 1;
3110014           }
3110015         else if (a[i] < b[i])
3110016           {
3110017             return -1;
3110018           }
3110019       }
3110020     return 0;
3110021 }
```

lib/string/memcpy.c

«

Si veda la sezione u0.70.

```
3120001 #include <string.h>
3120002 //---------------------------------------------------------------------
3120003 void *
3120004 memcpy (void *restrict dst, const void *restrict org, size_t n)
3120005 {
3120006     char *d = (char *) dst;
3120007     char *o = (char *) org;
3120008     size_t i;
3120009     for (i = 0; n > 0 && i < n; i++)
3120010       {
3120011         d[i] = o[i];
3120012       }
3120013     return dst;
3120014 }
```

lib/string/memmove.c

«

Si veda la sezione u0.71.

```
3130001 #include <string.h>
3130002 //---------------------------------------------------------------------
3130003 void *
3130004 memmove (void *dst, const void *org, size_t n)
3130005 {
```

```
3130006     char *d = (char *) dst;
3130007     char *o = (char *) org;
3130008     size_t i;
3130009     //
3130010     // Depending on the memory start locations, copy may be direct or
3130011     // reverse, to avoid overwriting before the relocation is done.
3130012     //
3130013     if (d < o)
3130014       {
3130015         for (i = 0; i < n; i++)
3130016           {
3130017             d[i] = o[i];
3130018           }
3130019       }
3130020     else if (d == o)
3130021       {
3130022         //
3130023         // Memory locations are already the same.
3130024         //
3130025         ;
3130026       }
3130027     else
3130028       {
3130029         for (i = n - 1; i >= 0; i--)
3130030           {
3130031             d[i] = o[i];
3130032           }
3130033       }
3130034     return dst;
3130035 }
```

lib/string/memset.c

«

Si veda la sezione u0.72.

```
3140001 #include <string.h>
3140002 //-----------------------------------------------------------------------
3140003 void *
3140004 memset (void *memory, int c, size_t n)
3140005 {
3140006     char  *m = (char *) memory;
3140007     size_t i;
3140008     for (i = 0; n > 0 && i < n; i++)
3140009       {
3140010         m[i] = (char) c;
3140011       }
3140012     return memory;
3140013 }
```

lib/string/strcat.c

«

Si veda la sezione u0.104.

```
3150001 #include <string.h>
3150002 //-----------------------------------------------------------------------
3150003 char *
3150004 strcat (char *restrict dst, const char *restrict org)
3150005 {
3150006     size_t i;
3150007     size_t j;
3150008     for (i = 0; dst[i] != 0; i++)
3150009       {
3150010         ; // Just look for the null character.
3150011       }
3150012     for (j = 0; org[j] != 0; i++, j++)
3150013       {
3150014         dst[i] = org[j];
3150015       }
3150016     dst[i] = 0;
3150017     return dst;
3150018 }
```

lib/string/strchr.c

«

Si veda la sezione u0.105.

```
3160001 #include <string.h>
3160002 //-----------------------------------------------------------------------
3160003 char *
3160004 strchr (const char *string, int c)
3160005 {
3160006     size_t i;
3160007     for (i = 0; ; i++)
3160008       {
3160009         if (string[i] == (char) c)
3160010           {
3160011             return (char *) (string + i);
3160012           }
3160013         else if (string[i] == 0)
3160014           {
3160015             return NULL;
3160016           }
3160017       }
3160018 }
```


lib/string/strcmp.c

«

Si veda la sezione u0.106.

```
3170001 #include <string.h>
3170002 //-----------------------------------------------------------------------
3170003 int
3170004 strcmp (const char *string1, const char *string2)
3170005 {
3170006     char *a = (char *) string1;
3170007     char *b = (char *) string2;
3170008     size_t i;
3170009     for (i = 0; ; i++)
3170010       {
3170011         if (a[i] > b[i])
3170012           {
3170013             return 1;
3170014           }
3170015         else if (a[i] < b[i])
3170016           {
3170017             return -1;
3170018           }
3170019         else if (a[i] == 0 && b[i] == 0)
3170020           {
3170021             return 0;
3170022           }
3170023       }
3170024 }
```

lib/string/strcoll.c

«

Si veda la sezione u0.106.

```
3180001 #include <string.h>
3180002 //-----------------------------------------------------------------------
3180003 int
3180004 strcoll (const char *string1, const char *string2)
3180005 {
3180006     return (strcmp (string1, string2));
3180007 }
```

lib/string/strcpy.c

«

Si veda la sezione u0.108.

```
3190001 #include <string.h>
3190002 //-----------------------------------------------------------------------
3190003 char *
3190004 strcpy (char *restrict dst, const char *restrict org)
3190005 {
3190006     size_t i;
3190007     for (i = 0; org[i] != 0; i++)
3190008       {
3190009         dst[i] = org[i];
3190010       }
3190011     dst[i] = 0;
3190012     return dst;
3190013 }
```

lib/string/strcspn.c

«

Si veda la sezione u0.118.

```
3200001 #include <string.h>
3200002 //-----------------------------------------------------------------------
3200003 size_t
3200004 strcspn (const char *string, const char *reject)
3200005 {
3200006     size_t i;
3200007     size_t j;
3200008     int found;
3200009     for (i = 0; string[i] != 0; i++)
3200010       {
3200011         for (j = 0, found = 0; reject[j] != 0 || found; j++)
3200012           {
3200013             if (string[i] == reject[j])
3200014               {
3200015                 found = 1;
3200016                 break;
3200017               }
3200018           }
3200019         if (found)
3200020           {
3200021             break;
3200022           }
3200023       }
3200024     return i;
3200025 }
```

lib/string/strdup.c

«

Si veda la sezione u0.110.

```
3210001 #include <string.h>
3210002 #include <stdlib.h>
3210003 #include <errno.h>
3210004 //-----------------------------------------------------------------------
3210005 char *
```

```
3210006 strdup  (const char *string)
3210007 {
3210008     size_t size;
3210009     char  *copy;
3210010     //
3210011     // Get string size: must be added 1, to count the termination null
3210012     // character.
3210013     //
3210014     size = strlen (string) + 1;
3210015     //
3210016     copy = malloc (size);
3210017     //
3210018     if (copy == NULL)
3210019       {
3210020         errset (ENOMEM);        // Not enough memory.
3210021         return (NULL);
3210022       }
3210023     //
3210024     strcpy (copy, string);
3210025     //
3210026     return (copy);
3210027 }
```

lib/string/strerror.c

«

Si veda la sezione u0.111.

```
3220001 #include <string.h>
3220002 #include <errno.h>
3220003 //-----------------------------------------------------------------------
3220004 #define ERROR_MAX 100
3220005 //-----------------------------------------------------------------------
3220006 char *
3220007 strerror (int errnum)
3220008 {
3220009     static char *err[ERROR_MAX];
3220010     //
3220011     err[0]                          = "No error";
3220012     err[E2BIG]                      = TEXT_E2BIG;
3220013     err[EACCES]                     = TEXT_EACCES;
3220014     err[EADDRINUSE]                 = TEXT_EADDRINUSE;
3220015     err[EADDRNOTAVAIL]              = TEXT_EADDRNOTAVAIL;
3220016     err[EAFNOSUPPORT]               = TEXT_EAFNOSUPPORT;
3220017     err[EAGAIN]                     = TEXT_EAGAIN;
3220018     err[EALREADY]                   = TEXT_EALREADY;
3220019     err[EBADF]                      = TEXT_EBADF;
3220020     err[EBADMSG]                    = TEXT_EBADMSG;
3220021     err[EBUSY]                      = TEXT_EBUSY;
3220022     err[ECANCELED]                  = TEXT_ECANCELED;
3220023     err[ECHILD]                     = TEXT_ECHILD;
3220024     err[ECONNABORTED]               = TEXT_ECONNABORTED;
3220025     err[ECONNREFUSED]               = TEXT_ECONNREFUSED;
3220026     err[ECONNRESET]                 = TEXT_ECONNRESET;
3220027     err[EDEADLK]                    = TEXT_EDEADLK;
3220028     err[EDESTADDRREQ]               = TEXT_EDESTADDRREQ;
3220029     err[EDOM]                       = TEXT_EDOM;
3220030     err[EDQUOT]                     = TEXT_EDQUOT;
3220031     err[EEXIST]                     = TEXT_EEXIST;
3220032     err[EFAULT]                     = TEXT_EFAULT;
3220033     err[EFBIG]                      = TEXT_EFBIG;
3220034     err[EHOSTUNREACH]               = TEXT_EHOSTUNREACH;
3220035     err[EIDRM]                      = TEXT_EIDRM;
3220036     err[EILSEQ]                     = TEXT_EILSEQ;
3220037     err[EINPROGRESS]                = TEXT_EINPROGRESS;
3220038     err[EINTR]                      = TEXT_EINTR;
3220039     err[EINVAL]                     = TEXT_EINVAL;
3220040     err[EIO]                        = TEXT_EIO;
3220041     err[EISCONN]                    = TEXT_EISCONN;
3220042     err[EISDIR]                     = TEXT_EISDIR;
3220043     err[ELOOP]                      = TEXT_ELOOP;
3220044     err[EMFILE]                     = TEXT_EMFILE;
3220045     err[EMLINK]                     = TEXT_EMLINK;
3220046     err[EMSGSIZE]                   = TEXT_EMSGSIZE;
3220047     err[EMULTIHOP]                  = TEXT_EMULTIHOP;
3220048     err[ENAMETOOLONG]               = TEXT_ENAMETOOLONG;
3220049     err[ENETDOWN]                   = TEXT_ENETDOWN;
3220050     err[ENETRESET]                  = TEXT_ENETRESET;
3220051     err[ENETUNREACH]                = TEXT_ENETUNREACH;
3220052     err[ENFILE]                     = TEXT_ENFILE;
3220053     err[ENOBUFS]                    = TEXT_ENOBUFS;
3220054     err[ENODATA]                    = TEXT_ENODATA;
3220055     err[ENODEV]                     = TEXT_ENODEV;
3220056     err[ENOENT]                     = TEXT_ENOENT;
3220057     err[ENOEXEC]                    = TEXT_ENOEXEC;
3220058     err[ENOLCK]                     = TEXT_ENOLCK;
3220059     err[ENOLINK]                    = TEXT_ENOLINK;
3220060     err[ENOMEM]                     = TEXT_ENOMEM;
3220061     err[ENOMSG]                     = TEXT_ENOMSG;
3220062     err[ENOPROTOOPT]                = TEXT_ENOPROTOOPT;
3220063     err[ENOSPC]                     = TEXT_ENOSPC;
3220064     err[ENOSR]                      = TEXT_ENOSR;
3220065     err[ENOSTR]                     = TEXT_ENOSTR;
3220066     err[ENOSYS]                     = TEXT_ENOSYS;
3220067     err[ENOTCONN]                   = TEXT_ENOTCONN;
3220068     err[ENOTDIR]                    = TEXT_ENOTDIR;
3220069     err[ENOTEMPTY]                  = TEXT_ENOTEMPTY;
3220070     err[ENOTSOCK]                   = TEXT_ENOTSOCK;
3220071     err[ENOTSUP]                    = TEXT_ENOTSUP;
3220072     err[ENOTTY]                     = TEXT_ENOTTY;
3220073     err[ENXIO]                      = TEXT_ENXIO;
3220074     err[EOPNOTSUPP]                 = TEXT_EOPNOTSUPP;
```



```
3220075     err[EOVERFLOW]                  = TEXT_EOVERFLOW;
3220076     err[EPERM]                      = TEXT_EPERM;
3220077     err[EPIPE]                      = TEXT_EPIPE;
3220078     err[EPROTO]                     = TEXT_EPROTO;
3220079     err[EPROTONOSUPPORT]            = TEXT_EPROTONOSUPPORT;
3220080     err[EPROTOTYPE]                 = TEXT_EPROTOTYPE;
3220081     err[ERANGE]                     = TEXT_ERANGE;
3220082     err[EROFS]                      = TEXT_EROFS;
3220083     err[ESPIPE]                     = TEXT_ESPIPE;
3220084     err[ESRCH]                      = TEXT_ESRCH;
3220085     err[ESTALE]                     = TEXT_ESTALE;
3220086     err[ETIME]                      = TEXT_ETIME;
3220087     err[ETIMEDOUT]                  = TEXT_ETIMEDOUT;
3220088     err[ETXTBSY]                    = TEXT_ETXTBSY;
3220089     err[EWOULDBLOCK]                = TEXT_EWOULDBLOCK;
3220090     err[EXDEV]                      = TEXT_EXDEV;
3220091     err[E_FILE_TYPE]                = TEXT_E_FILE_TYPE;
3220092     err[E_ROOT_INODE_NOT_CACHED]    = TEXT_E_ROOT_INODE_NOT_CACHED;
3220093     err[E_CANNOT_READ_SUPERBLOCK]   = TEXT_E_CANNOT_READ_SUPERBLOCK;
3220094     err[E_MAP_INODE_TOO_BIG]        = TEXT_E_MAP_INODE_TOO_BIG;
3220095     err[E_MAP_ZONE_TOO_BIG]         = TEXT_E_MAP_ZONE_TOO_BIG;
3220096     err[E_DATA_ZONE_TOO_BIG]        = TEXT_E_DATA_ZONE_TOO_BIG;
3220097     err[E_CANNOT_FIND_ROOT_DEVICE]  = TEXT_E_CANNOT_FIND_ROOT_DEVICE;
3220098     err[E_CANNOT_FIND_ROOT_INODE]   = TEXT_E_CANNOT_FIND_ROOT_INODE;
3220099     err[E_FILE_TYPE_UNSUPPORTED]    = TEXT_E_FILE_TYPE_UNSUPPORTED;
3220100     err[E_ENV_TOO_BIG]              = TEXT_E_ENV_TOO_BIG;
3220101     err[E_LIMIT]                    = TEXT_E_LIMIT;
3220102     err[E_NOT_MOUNTED]              = TEXT_E_NOT_MOUNTED;
3220103     err[E_NOT_IMPLEMENTED]          = TEXT_E_NOT_IMPLEMENTED;
3220104     //
3220105     if (errnum >= ERROR_MAX || errnum < 0)
3220106       {
3220107         return ("Unknown error");
3220108       }
3220109     //
3220110     return (err[errnum]);
3220111 }
```

lib/string/strlen.c

«

Si veda la sezione u0.112.

```
3230001 #include <string.h>
3230002 //-----------------------------------------------------------------------
3230003 size_t
3230004 strlen (const char *string)
3230005 {
3230006     size_t i;
3230007     for (i = 0; string[i] != 0 ; i++)
3230008       {
3230009         ; // Just count.
3230010       }
3230011     return i;
3230012 }
```

lib/string/strncat.c

«

Si veda la sezione u0.104.

```
3240001 #include <string.h>
3240002 //-----------------------------------------------------------------------
3240003 char *
3240004 strncat (char *restrict dst, const char *restrict org, size_t n)
3240005 {
3240006     size_t i;
3240007     size_t j;
3240008     for (i = 0; n > 0 && dst[i] != 0; i++)
3240009       {
3240010         ; // Just seek the null character.
3240011       }
3240012     for (j = 0; n > 0 && j < n && org[j] != 0; i++, j++)
3240013       {
3240014         dst[i] = org[j];
3240015       }
3240016     dst[i] = 0;
3240017     return dst;
3240018 }
```

lib/string/strncmp.c

«

Si veda la sezione u0.106.

```
3250001 #include <string.h>
3250002 //-----------------------------------------------------------------------
3250003 int
3250004 strncmp (const char *string1, const char *string2, size_t n)
3250005 {
3250006     size_t i;
3250007     for (i = 0; i < n ; i++)
3250008       {
3250009         if (string1[i] > string2[i])
3250010           {
3250011             return 1;
3250012           }
3250013         else if (string1[i] < string2[i])
3250014           {
3250015             return -1;
3250016           }
```

```
3250017         else if (string1[i] == 0 && string2[i] == 0)
3250018           {
3250019             return 0;
3250020           }
3250021       }
3250022     return 0;
3250023 }
```

lib/string/strncpy.c

«

Si veda la sezione u0.108.

```
3260001 #include <string.h>
3260002 //---------------------------------------------------------------------
3260003 char *
3260004 strncpy (char *restrict dst, const char *restrict org, size_t n)
3260005 {
3260006     size_t i;
3260007     for (i = 0; n > 0 && i < n && org[i] != 0; i++)
3260008       {
3260009         dst[i] = org[i];
3260010       }
3260011     for ( ; n > 0 && i < n; i++)
3260012       {
3260013         dst[i] = 0;
3260014       }
3260015     return dst;
3260016 }
```

lib/string/strpbrk.c

«

Si veda la sezione u0.116.

```
3270001 #include <string.h>
3270002 //---------------------------------------------------------------------
3270003 char *
3270004 strpbrk (const char *string, const char *accept)
3270005 {
3270006     size_t i;
3270007     size_t j;
3270008     for (i = 0; string[i] != 0; i++)
3270009       {
3270010         for (j = 0; accept[j] != 0; j++)
3270011           {
3270012             if (string[i] == accept[j])
3270013               {
3270014                 return (string + i);
3270015               }
3270016           }
3270017       }
3270018     return NULL;
3270019 }
```

lib/string/strrchr.c

«

Si veda la sezione u0.105.

```
3280001 #include <string.h>
3280002 //---------------------------------------------------------------------
3280003 char *
3280004 strrchr (const char *string, int c)
3280005 {
3280006     int i;
3280007     for (i = strlen (string); i >= 0; i--)
3280008       {
3280009         if (string[i] == (char) c)
3280010           {
3280011             break;
3280012           }
3280013       }
3280014     if (i < 0)
3280015       {
3280016         return NULL;
3280017       }
3280018     else
3280019       {
3280020         return (string + i);
3280021       }
3280022 }
```

lib/string/strspn.c

«

Si veda la sezione u0.118.

```
3290001 #include <string.h>
3290002 //---------------------------------------------------------------------
3290003 size_t
3290004 strspn (const char *string, const char *accept)
3290005 {
3290006     size_t i;
3290007     size_t j;
3290008     int found;
3290009     for (i = 0; string[i] != 0; i++)
3290010       {
3290011         for (j = 0, found = 0; accept[j] != 0; j++)
3290012           {
3290013             if (string[i] == accept[j])
```



```
3290014               {
3290015                 found = 1;
3290016                 break;
3290017               }
3290018           }
3290019         if (!found)
3290020           {
3290021             break;
3290022           }
3290023       }
3290024     return i;
3290025 }
```

lib/string/strstr.c

«

Si veda la sezione u0.119.

```
3300001 #include <string.h>
3300002 //---------------------------------------------------------------------
3300003 char *
3300004 strstr (const char *string, const char *substring)
3300005 {
3300006     size_t i;
3300007     size_t j;
3300008     size_t k;
3300009     int found;
3300010     if (substring[0] == 0)
3300011       {
3300012         return (char *) string;
3300013       }
3300014     for (i = 0, j = 0, found = 0; string[i] != 0; i++)
3300015       {
3300016         if (string[i] == substring[0])
3300017           {
3300018             for (k = i, j = 0;
3300019                  string[k] == substring[j] &&
3300020                  string[k] != 0            &&
3300021                  substring[j] != 0;
3300022                  j++, k++)
3300023               {
3300024                 ;
3300025               }
3300026             if (substring[j] == 0)
3300027               {
3300028                 found = 1;
3300029               }
3300030           }
3300031         if (found)
3300032           {
3300033             return (char *) (string + i);
3300034           }
3300035       }
3300036     return NULL;
3300037 }
```

lib/string/strtok.c

«

Si veda la sezione u0.120.

```
3310001 #include <string.h>
3310002 //---------------------------------------------------------------------
3310003 char *
3310004 strtok (char *restrict string, const char *restrict delim)
3310005 {
3310006     static char *next = NULL;
3310007     size_t i = 0;
3310008     size_t j;
3310009     int found_token;
3310010     int found_delim;
3310011     //
3310012     // If the string received a the first parameter is a null pointer,
3310013     // the static pointer is used. But if it is already NULL,
3310014     // the scan cannot start.
3310015     //
3310016     if (string == NULL)
3310017       {
3310018         if (next == NULL)
3310019           {
3310020             return NULL;
3310021           }
3310022         else
3310023           {
3310024             string = next;
3310025           }
3310026       }
3310027     //
3310028     // If the string received as the first parameter is empty, the scan
3310029     // cannot start.
3310030     //
3310031     if (string[0] == 0)
3310032       {
3310033         next = NULL;
3310034         return NULL;
3310035       }
3310036     else
3310037       {
3310038         if (delim[0] == 0)
3310039           {
3310040             return string;
3310041           }
```

```
      }
    //
    // Find the next token.
    //
    for (i = 0, found_token = 0, j = 0;
         string[i] != 0 && (!found_token); i++)
      {
        //
        // Look inside delimiters.
        //
        for (j = 0, found_delim = 0; delim[j] != 0; j++)
          {
            if (string[i] == delim[j])
              {
                found_delim = 1;
              }
          }
        //
        // If current character inside the string is not a delimiter,
        // it is the start of a new token.
        //
        if (!found_delim)
          {
            found_token = 1;
            break;
          }
      }
    //
    // If a token was found, the pointer is updated.
    // If otherwise the token is not found, this means that
    // there are no more.
    //
    if (found_token)
      {
        string += i;
      }
    else
      {
        next = NULL;
        return NULL;
      }
    //
    // Find the end of the token.
    //
    for (i = 0, found_delim = 0; string[i] != 0; i++)
      {
        for (j = 0; delim[j] != 0; j++)
          {
            if (string[i] == delim[j])
              {
                found_delim = 1;
                break;
              }
          }
        if (found_delim)
          {
            break;
          }
      }
    //
    // If a delimiter was found, the corresponding character must be
    // reset to zero. If otherwise the string is terminated, the
    // scan is terminated.
    //
    if (found_delim)
      {
        string[i] = 0;
        next = &string[i+1];
      }
    else
      {
        next = NULL;
      }
    //
    // At this point, the current string represent the token found.
    //
    return string;
}
```

lib/string/strxfrm.c

«

Si veda la sezione u0.123.

```
#include <string.h>
//---------------------------------------------------------------------
size_t
strxfrm (char *restrict dst, const char *restrict org, size_t n)
{
    size_t i;
    if (n == 0 && dst == NULL)
      {
        return strlen (org);
      }
    else
      {
        for (i = 0; i < n ; i++)
          {
            dst[i] = org[i];
            if (org[i] == 0)
              {
                break;
              }
          }
        return i;
      }
}
```

os16: «lib/sys/os16.h»

«

Si veda la sezione u0.2.

```
#ifndef _SYS_OS16_H
#define _SYS_OS16_H      1
//---------------------------------------------------------------------
// This file contains all the declarations that don't have a better
// place inside standard headers files. Even declarations related to
// device numbers and system calls is contained here.
//---------------------------------------------------------------------
// Please remember that system calls should never be used (called)
// inside the kernel code, because system calls cannot be nested for
// the os16 simple architecture!
// If a particular function is necessary inside the kernel, that usually
// is made by a system call, an appropriate k_...() function must be
// made, to avoid the problem.
//---------------------------------------------------------------------

#include <sys/types.h>
#include <sys/stat.h>
#include <stdint.h>
#include <signal.h>
#include <limits.h>
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <const.h>
#include <restrict.h>
#include <stdarg.h>
//---------------------------------------------------------------------
// Device numbers.
//---------------------------------------------------------------------
#define DEV_UNDEFINED_MAJOR     0x00
#define DEV_UNDEFINED           0x0000
#define DEV_MEM_MAJOR           0x01
#define DEV_MEM                 0x0101
#define DEV_NULL                0x0102
#define DEV_PORT                0x0103
#define DEV_ZERO                0x0104
#define DEV_TTY_MAJOR           0x02
#define DEV_TTY                 0x0200
#define DEV_DSK_MAJOR           0x03
#define DEV_DSK0                0x0300
#define DEV_DSK1                0x0301
#define DEV_DSK2                0x0302
#define DEV_DSK3                0x0303
#define DEV_KMEM_MAJOR          0x04
#define DEV_KMEM_PS             0x0401
#define DEV_KMEM_MMP            0x0402
#define DEV_KMEM_SB             0x0403
#define DEV_KMEM_INODE          0x0404
#define DEV_KMEM_FILE           0x0405
#define DEV_CONSOLE_MAJOR       0x05
#define DEV_CONSOLE             0x05FF
#define DEV_CONSOLE0            0x0500
#define DEV_CONSOLE1            0x0501
#define DEV_CONSOLE2            0x0502
#define DEV_CONSOLE3            0x0503
#define DEV_CONSOLE4            0x0504
//---------------------------------------------------------------------
// Current segments.
//---------------------------------------------------------------------
uint16_t _seg_i (void);
uint16_t _seg_d (void);
uint16_t _cs    (void);
uint16_t _ds    (void);
uint16_t _ss    (void);
uint16_t _es    (void);
uint16_t _sp    (void);
uint16_t _bp    (void);
#define seg_i() ((unsigned int) _seg_i ())
#define seg_d() ((unsigned int) _seg_d ())
#define cs()    ((unsigned int) _cs ())
#define ds()    ((unsigned int) _ds ())
#define ss()    ((unsigned int) _ss ())
#define es()    ((unsigned int) _es ())
#define sp()    ((unsigned int) _sp ())
#define bp()    ((unsigned int) _bp ())
//---------------------------------------------------------------------
#define min(a, b) (a < b ? a : b)
#define max(a, b) (a > b ? a : b)
//---------------------------------------------------------------------
#define INPUT_LINE_HIDDEN 0
#define INPUT_LINE_ECHO   1
#define INPUT_LINE_STARS  2
//---------------------------------------------------------------------
#define MOUNT_DEFAULT     0  // Default mount options.
#define MOUNT_RO          1  // Read only mount option.
//---------------------------------------------------------------------
#define SYS_0                   0  // Nothing to do.
#define SYS_CHDIR               1
#define SYS_CHMOD               2
#define SYS_CLOCK               3
#define SYS_CLOSE               4
```

```
#define SYS_EXEC                5
#define SYS_EXIT                6  // [1] see below.
#define SYS_FCHMOD              7
#define SYS_FORK                8
#define SYS_FSTAT               9
#define SYS_KILL               10
#define SYS_LSEEK              11
#define SYS_MKDIR              12
#define SYS_MKNOD              13
#define SYS_MOUNT              14
#define SYS_OPEN               15
#define SYS_PGRP               16
#define SYS_READ               17
#define SYS_SETEUID            18
#define SYS_SETUID             19
#define SYS_SIGNAL             20
#define SYS_SLEEP              21
#define SYS_STAT               22
#define SYS_TIME               23
#define SYS_UAREA              24
#define SYS_UMASK              25
#define SYS_UMOUNT             26
#define SYS_WAIT               27
#define SYS_WRITE              28
#define SYS_ZPCHAR             29  // [2] see below.
#define SYS_ZPSTRING           30  // [2]
#define SYS_CHOWN              31
#define SYS_DUP                33
#define SYS_DUP2               34
#define SYS_LINK               35
#define SYS_UNLINK             36
#define SYS_FCNTL              37
#define SYS_STIME              38
#define SYS_FCHOWN             39
//
// [1] The files 'crt0...' need to know the value used for the
//     exit system call. If this value is modified, all the file
//     'crt0...' have also to be modified the same way.
//
// [2] These system calls were developed at the beginning, when no
//     standard I/O was available. They are to be considered as a
//     last resort for debugging purposes.
//
//------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_chdir_t;
//------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    mode_t       mode;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_chmod_t;
//------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    uid_t        uid;
    uid_t        gid;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_chown_t;
//------------------------------------------------------------------
typedef struct {
    clock_t      ret;
} sysmsg_clock_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_close_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn_old;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_dup_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn_old;
    int          fdn_new;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_dup2_t;
//------------------------------------------------------------------
typedef struct {
```



```
    char         path[PATH_MAX];
    int          argc;
    int          envc;
    char         arg_data[ARG_MAX/2];
    char         env_data[ARG_MAX/2];
    uid_t        uid;
    uid_t        euid;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_exec_t;
//------------------------------------------------------------------
typedef struct {
    int          status;
} sysmsg_exit_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    mode_t       mode;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_fchmod_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    uid_t        uid;
    uid_t        gid;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_fchown_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    int          cmd;
    int          arg;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_fcntl_t;
//------------------------------------------------------------------
typedef struct {
    pid_t        ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_fork_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    struct stat  stat;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_fstat_t;
//------------------------------------------------------------------
typedef struct {
    pid_t        pid;
    int          signal;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_kill_t;
//------------------------------------------------------------------
typedef struct {
    char         path_old[PATH_MAX];
    char         path_new[PATH_MAX];
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_link_t;
//------------------------------------------------------------------
typedef struct {
    int          fdn;
    off_t        offset;
    int          whence;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_lseek_t;
//------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    mode_t       mode;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_mkdir_t;
//------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
```

```
    mode_t       mode;
    dev_t        device;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_mknod_t;
//---------------------------------------------------------------------
typedef struct {
    char         path_dev[PATH_MAX];
    char         path_mnt[PATH_MAX];
    int          options;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_mount_t;
//---------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    int          flags;
    mode_t       mode;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_open_t;
//---------------------------------------------------------------------
typedef struct {
    int          fdn;
    char         buffer[BUFSIZ];
    size_t       count;
    int          eof;
    ssize_t      ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_read_t;
//---------------------------------------------------------------------
typedef struct {
    uid_t        euid;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_seteuid_t;
//---------------------------------------------------------------------
typedef struct {
    uid_t        uid;
    uid_t        euid;
    uid_t        suid;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_setuid_t;
//---------------------------------------------------------------------
typedef struct {
    sighandler_t handler;
    int          signal;
    sighandler_t ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_signal_t;
//---------------------------------------------------------------------
#define WAKEUP_EVENT_SIGNAL  1    // 1, 2, 4, 8, 16,...
#define WAKEUP_EVENT_TIMER   2    // so that can be 'OR' combined.
#define WAKEUP_EVENT_TTY     4    //
typedef struct {
    int          events;
    int          signal;
    unsigned int seconds;
    time_t       ret;
} sysmsg_sleep_t;
//---------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    struct stat  stat;
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_stat_t;
//---------------------------------------------------------------------
typedef struct {
    time_t       ret;
} sysmsg_time_t;
//---------------------------------------------------------------------
typedef struct {
    time_t       timer;
    int          ret;
} sysmsg_stime_t;
//---------------------------------------------------------------------
typedef struct {
    uid_t        uid;         // Read user ID.
    uid_t        euid;        // Effective user ID.
    uid_t        suid;        // Saved user ID.
    pid_t        pid;         // Process ID.
    pid_t        ppid;        // Parent PID.
    pid_t        pgrp;        // Process group.
```



```
    mode_t       umask;       // Access permission mask.
    char         path_cwd[PATH_MAX];
} sysmsg_uarea_t;
//---------------------------------------------------------------------
typedef struct {
    mode_t       umask;
    mode_t       ret;
} sysmsg_umask_t;
//---------------------------------------------------------------------
typedef struct {
    char         path_mnt[PATH_MAX];
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_umount_t;
//---------------------------------------------------------------------
typedef struct {
    char         path[PATH_MAX];
    int          ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_unlink_t;
//---------------------------------------------------------------------
typedef struct {
    int          status;
    pid_t        ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_wait_t;
//---------------------------------------------------------------------
typedef struct {
    int          fdn;
    char         buffer[BUFSIZ];
    size_t       count;
    ssize_t      ret;
    int          errno;
    int          errln;
    char         errfn[PATH_MAX];
} sysmsg_write_t;
//---------------------------------------------------------------------
typedef struct {
    char         c;
} sysmsg_zpchar_t;
//---------------------------------------------------------------------
typedef struct {
    char         string[BUFSIZ];
} sysmsg_zpstring_t;
//---------------------------------------------------------------------
void  heap_clear   (void);
int   heap_min     (void);
void  input_line   (char *line, char *prompt, size_t size, int type);
int   mount        (const char *path_dev, const char *path_mnt,
                    int options);
int   namep        (const char *name, char *path, size_t size);
void  process_info (void);
void  sys          (int syscallnr, void *message, size_t size);
int   umount       (const char *path_mnt);
void  z_perror     (const char *string);
int   z_printf     (const char *restrict format, ...);
int   z_putchar    (int c);
int   z_puts       (const char *string);
int   z_vprintf    (const char *restrict format, va_list arg);
//int  z_vsprintf   (char *restrict string, const char *restrict format,
//                   va_list arg);
//---------------------------------------------------------------------

#endif
```

lib/sys/os16/_bp.s

Si veda la sezione u0.12.

```
.global __bp
.text
;---------------------------------------------------------------------
; Read the base pointer, as it is before this call.
;---------------------------------------------------------------------
.align 2
__bp:
    enter #2, #0        ; 1 local variable.
    pushf
    cli
    pusha
    mov   ax, [bp]      ; The previous BP value is saved at *BP.
    mov   -2[bp], ax    ; Save the calculated old SP value.
    popa
    popf
    mov   ax, -2[bp]    ; AX is the function return value.
    leave
    ret
```

lib/sys/os16/_cs.s

«

Si veda la sezione u0.12.

```
3350001  .global __cs
3350002  .text
3350003  ;--------------------------------------------------------------------------
3350004  ; Read the code segment value.
3350005  ;--------------------------------------------------------------------------
3350006  .align 2
3350007  __cs:
3350008      mov ax, cs
3350009      ret
```

lib/sys/os16/_ds.s

«

Si veda la sezione u0.12.

```
3360001  .global __ds
3360002  .text
3360003  ;--------------------------------------------------------------------------
3360004  ; Read the data segment value.
3360005  ;--------------------------------------------------------------------------
3360006  .align 2
3360007  __ds:
3360008      mov ax, ds
3360009      ret
```

lib/sys/os16/_es.s

«

Si veda la sezione u0.12.

```
3370001  .global __es
3370002  .text
3370003  ;--------------------------------------------------------------------------
3370004  ; Read the extra segment value.
3370005  ;--------------------------------------------------------------------------
3370006  .align 2
3370007  __es:
3370008      mov ax, es
3370009      ret
```

lib/sys/os16/_seg_d.s

«

Si veda la sezione u0.91.

```
3380001  .global __seg_d
3380002  .text
3380003  ;--------------------------------------------------------------------------
3380004  ; Read the data segment value.
3380005  ;--------------------------------------------------------------------------
3380006  .align 2
3380007  __seg_d:
3380008      mov ax, ds
3380009      ret
```

lib/sys/os16/_seg_i.s

«

Si veda la sezione u0.91.

```
3390001  .global __seg_i
3390002  .text
3390003  ;--------------------------------------------------------------------------
3390004  ; Read the instruction segment value.
3390005  ;--------------------------------------------------------------------------
3390006  .align 2
3390007  __seg_i:
3390008      mov ax, cs
3390009      ret
```

lib/sys/os16/_sp.s

«

Si veda la sezione u0.12.

```
3400001  .global __sp
3400002  .text
3400003  ;--------------------------------------------------------------------------
3400004  ; Read the stack pointer, as it is before this call.
3400005  ;--------------------------------------------------------------------------
3400006  .align 2
3400007  __sp:
3400008      enter #2, #0        ; 1 local variable.
3400009      pushf
3400010      cli
3400011      pusha
3400012      mov   ax, bp        ; The previous SP is equal to BP + 2 + 2.
3400013      add   ax, #4        ;
3400014      mov   -2[bp], ax    ; Save the calculated old SP value.
3400015      popa
3400016      popf
3400017      mov   ax, -2[bp]    ; AX is the function return value.
3400018      leave
3400019      ret
```


lib/sys/os16/_ss.s

«

Si veda la sezione u0.12.

```
3410001  .global __ss
3410002  .text
3410003  ;--------------------------------------------------------------------------
3410004  ; Read the stack segment value.
3410005  ;--------------------------------------------------------------------------
3410006  .align 2
3410007  __ss:
3410008      mov ax, ss
3410009      ret
```

lib/sys/os16/heap_clear.c

«

Si veda la sezione u0.57.

```
3420001  #include <sys/os16.h>
3420002  //--------------------------------------------------------------------------
3420003  extern uint16_t _end;
3420004  //--------------------------------------------------------------------------
3420005  void heap_clear (void)
3420006  {
3420007      uint16_t *a = &_end;
3420008      uint16_t *z = (void *) (sp () - 2);
3420009      for (; a < z; a++)
3420010        {
3420011          *a = 0xFFFF;
3420012        }
3420013  }
```

lib/sys/os16/heap_min.c

«

Si veda la sezione u0.57.

```
3430001  #include <sys/os16.h>
3430002  //--------------------------------------------------------------------------
3430003  extern uint16_t _end;
3430004  //--------------------------------------------------------------------------
3430005  int heap_min (void)
3430006  {
3430007      uint16_t *a = &_end;
3430008      uint16_t *z = (void *) (sp () - 2);
3430009      int      count;
3430010      for (count = 0; a < z && *a == 0xFFFF; a++, count++);
3430011      return (count * 2);
3430012  }
```

lib/sys/os16/input_line.c

«

Si veda la sezione u0.60.

```
3440001  #include <sys/os16.h>
3440002  #include <string.h>
3440003  #include <stdio.h>
3440004  //--------------------------------------------------------------------------
3440005  void
3440006  input_line (char *line, char *prompt, size_t size, int type)
3440007  {
3440008      int   i;    // Index inside the 'line[]' array.
3440009      int   c;    // Character received from keyboard.
3440010
3440011      if (prompt != NULL || strlen (prompt) > 0)
3440012        {
3440013          printf ("%s ", prompt);
3440014        }
3440015      //
3440016      // Loop for character input. Please note that the loop
3440017      // will exit only through 'break', where the input line
3440018      // will also be correctly terminated with '\0'.
3440019      //
3440020      for (i = 0;; i++)
3440021        {
3440022          c = getchar ();
3440023          //
3440024          // Control codes.
3440025          //
3440026          if (c == EOF)
3440027            {
3440028              line[i] = 0;
3440029              break;
3440030            }
3440031          else if (c == 4)        // [Ctrl D]
3440032            {
3440033              line[i] = 0;
3440034              break;
3440035            }
3440036          else if (c == 10)       // [Enter]
3440037            {
3440038              line[i] = 0;
3440039              break;
3440040            }
3440041          else if (c == 8)        // [Backspace]
3440042            {
3440043              if (i == 0)
3440044                {
3440045                  //
3440046                  // It is already the lowest position, so the video
```

```
                // cursor is moved forward again, so that the prompt
                // is not overwritten.
                // The index is set to -1, so that on the next loop,
                // it will be again zero.
                //
                printf (" ");
                i = -1;
              }
            else
              {
                i -= 2;
              }
            continue;
          }
        //
        // If 'i' is equal 'size - 1', it is not allowed to continue
        // typing.
        //
        if (i == (size - 1))
          {
            //
            // Ignore typing, move back the cursor, delete the character
            // typed and move back again.
            //
            printf ("\b \b");
            i--;
            continue;
          }
        //
        // Typing is allowed.
        //
        line[i] = c;
        //
        // Verify if it should be hidden.
        //
        if (type == INPUT_LINE_HIDDEN)
          {
            printf ("\b ");     // Space: at least you see something.
          }
        else if (type == INPUT_LINE_STARS)
          {
            printf ("\b*");
          }
      }
}
```

lib/sys/os16/mount.c

« Si veda la sezione u0.27.

```
#include <sys/types.h>
#include <errno.h>
#include <sys/os16.h>
#include <stddef.h>
#include <string.h>
#include <const.h>
//---------------------------------------------------------------------------
int
mount (const char *path_dev, const char *path_mnt, int options)
{
    sysmsg_mount_t msg;
    //
    strncpy (msg.path_dev, path_dev, PATH_MAX);
    strncpy (msg.path_mnt, path_mnt, PATH_MAX);
    msg.options = options;
    msg.ret     = 0;
    msg.errno   = 0;
    //
    sys (SYS_MOUNT, &msg, (sizeof msg));
    //
    errno = msg.errno;
    errln = msg.errln;
    strncpy (errfn, msg.errfn, PATH_MAX);
    return (msg.ret);
}
```

lib/sys/os16/namep.c

« Si veda la sezione u0.74.

```
#include <sys/os16.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
//---------------------------------------------------------------------------
int
namep (const char *name, char *path, size_t size)
{
    char  command[PATH_MAX];
    char *env_path;
    int   p;            // Index used inside the path environment.
    int   c;            // Index used inside the command string.
    int   status;
    //
    // Check for valid input.
    //
    if (name == NULL || name[0] == 0 || path == NULL || name == path)
      {
        errset (EINVAL);                // Invalid argument.
        return (-1);
```



```
      }
    //
    // Check if the original command contains at least a '/'. Otherwise
    // a scan for the environment variable 'PATH' must be done.
    //
    if (strchr (name, '/') == NULL)
      {
        //
        // Ok: no '/' there. Get the environment variable 'PATH'.
        //
        env_path = getenv ("PATH");
        if (env_path == NULL)
          {
            //
            // There is no 'PATH' environment value.
            //
            errset (ENOENT);            // No such file or directory.
            return (-1);
          }
        //
        // Scan paths and try to find a file with that name.
        //
        for (p = 0; env_path[p] != 0;)
          {
            for (c = 0;
                 c < (PATH_MAX - strlen(name) - 2) &&
                 env_path[p] != 0                  &&
                 env_path[p] != ':';
                 c++, p++)
              {
                command[c] = env_path[p];
              }
            //
            // If the loop is ended because the command array does not
            // have enough room for the full path, then must return an
            // error.
            //
            if (env_path[p] != ':' && env_path[p] != 0)
              {
                errset (ENAMETOOLONG);  // Filename too long.
                return (-1);
              }
            //
            // The command array has enough space. At index 'c' must
            // place a zero, to terminate current string.
            //
            command[c] = 0;
            //
            // Add the rest of the path.
            //
            strcat (command, "/");
            strcat (command, name);
            //
            // Verify to have something with that full path name.
            //
            status = access (command, F_OK);
            if (status == 0)
              {
                //
                // Verify to have enough room inside the destination
                // path.
                //
                if (strlen (command) >= size)
                  {
                    //
                    // Sorry: too big. There must be room also for
                    // the string termination null character.
                    //
                    errset (ENAMETOOLONG);      // Filename too long.
                    return (-1);
                  }
                //
                // Copy the path and return.
                //
                strncpy (path, command, size);
                return (0);
              }
            //
            // That path was not good: try again. But before returning
            // to the external loop, must verify if 'p' is to be
            // incremented, after a ':', because the external loop
            // does not touch the index 'p',
            //
            if (env_path[p] == ':')
              {
                p++;
              }
          }
      }
    //
    // At this point, there is no match with the paths.
    //
    errset (ENOENT);                // No such file or directory.
    return (-1);
  }
//
// At this point, a path was given and the environment variable
// 'PATH' was not scanned. Just copy the same path. But must verify
// that the receiving path has enough room for it.
//
if (strlen (name) >= size)
  {
```

```
3460122         //
3460123         // Sorry: too big.
3460124         //
3460125         errset (ENAMETOOLONG);  // Filename too long.
3460126         return (-1);
3460127       }
3460128     //
3460129     // Ok: copy and return.
3460130     //
3460131     strncpy (path, name, size);
3460132     return (0);
3460133 }
```

lib/sys/os16/process_info.c

«

Si veda la sezione u0.79.

```
3470001 #include <sys/os16.h>
3470002 #include <stdio.h>
3470003
3470004 extern uint16_t _edata;
3470005 extern uint16_t _end;
3470006 //---------------------------------------------------------------------
3470007 void
3470008 process_info (void)
3470009 {
3470010     printf ("cs=%04x ds=%04x ss=%04x es=%04x bp=%04x sp=%04x ",
3470011             cs (), ds (), ss (), es (), bp (), sp ());
3470012     printf ("edata=%04x ebss=%04x heap=%04x\n",
3470013             (int) &_edata, (int) &_end, heap_min ());
3470014 }
```

lib/sys/os16/sys.s

«

Si veda la sezione u0.37.

```
3480001 .global _sys
3480002 .text
3480003 ;---------------------------------------------------------------------
3480004 ; Call a system call.
3480005 ;
3480006 ; Please remember that system calls should never be used (called) inside
3480007 ; the kernel code, because system calls cannot be nested for the os16
3480008 ; simple architecture!
3480009 ; If a particular function is necessary inside the kernel, that usually
3480010 ; is made by a system call, an appropriate k_...() function must be
3480011 ; made, to avoid the problem.
3480012 ;
3480013 ;---------------------------------------------------------------------
3480014 .align 2
3480015 _sys:
3480016     int   #0x80
3480017     ret
```

lib/sys/os16/umount.c

«

Si veda la sezione u0.27.

```
3490001 #include <sys/types.h>
3490002 #include <errno.h>
3490003 #include <sys/os16.h>
3490004 #include <stddef.h>
3490005 #include <string.h>
3490006 //---------------------------------------------------------------------
3490007 int
3490008 umount (const char *path_mnt)
3490009 {
3490010     sysmsg_umount_t msg;
3490011     //
3490012     strncpy (msg.path_mnt, path_mnt, PATH_MAX);
3490013     msg.ret     = 0;
3490014     msg.errno   = 0;
3490015     //
3490016     sys (SYS_UMOUNT, &msg, (sizeof msg));
3490017     //
3490018     errno = msg.errno;
3490019     errln = msg.errln;
3490020     strncpy (errfn, msg.errfn, PATH_MAX);
3490021     return (msg.ret);
3490022 }
```

lib/sys/os16/z_perror.c

«

Si veda la sezione u0.45.

```
3500001 #include <sys/os16.h>
3500002 #include <errno.h>
3500003 #include <stddef.h>
3500004 #include <string.h>
3500005 //---------------------------------------------------------------------
3500006 void
3500007 z_perror (const char *string)
3500008 {
3500009     //
3500010     // If errno is zero, there is nothing to show.
3500011     //
3500012     if (errno == 0)
```



```
3500013       {
3500014         return;
3500015       }
3500016     //
3500017     // Show the string if there is one.
3500018     //
3500019     if (string != NULL && strlen (string) > 0)
3500020       {
3500021         z_printf ("%s: ", string);
3500022       }
3500023     //
3500024     // Show the translated error.
3500025     //
3500026     if (errfn[0] != 0 && errln != 0)
3500027       {
3500028         z_printf ("[%s:%u:%i] %s\n",
3500029                   errfn, errln, errno, strerror (errno));
3500030       }
3500031     else
3500032       {
3500033         z_printf ("[%i] %s\n", errno, strerror (errno));
3500034       }
3500035 }
```

lib/sys/os16/z_printf.c

«

Si veda la sezione u0.45.

```
3510001 #include <sys/os16.h>
3510002 //---------------------------------------------------------------------
3510003 int
3510004 z_printf (char *format, ...)
3510005 {
3510006     va_list ap;
3510007     va_start (ap, format);
3510008     return z_vprintf (format, ap);
3510009 }
```

lib/sys/os16/z_putchar.c

«

Si veda la sezione u0.45.

```
3520001 #include <sys/os16.h>
3520002 //---------------------------------------------------------------------
3520003 int
3520004 z_putchar (int c)
3520005 {
3520006     sysmsg_zpchar_t msg;
3520007     msg.c = c;
3520008     sys (SYS_ZPCHAR, &msg, (sizeof msg));
3520009     return (c);
3520010 }
```

lib/sys/os16/z_puts.c

«

Si veda la sezione u0.45.

```
3530001 #include <sys/os16.h>
3530002 //---------------------------------------------------------------------
3530003 int
3530004 z_puts (char *string)
3530005 {
3530006     unsigned int i;
3530007     for (i = 0; string[i] != 0; string++)
3530008       {
3530009         z_putchar ((int) string[i]);
3530010       }
3530011     z_putchar ((int) '\n');
3530012     return (1);
3530013 }
```

lib/sys/os16/z_vprintf.c

«

Si veda la sezione u0.45.

```
3540001 #include <sys/os16.h>
3540002 //---------------------------------------------------------------------
3540003 int
3540004 z_vprintf (char *format, va_list arg)
3540005 {
3540006     int                         ret;
3540007     sysmsg_zpstring_t msg;
3540008     msg.string[0] = 0;
3540009     ret = vsprintf (msg.string, format, arg);
3540010     sys (SYS_ZPSTRING, &msg, (sizeof msg));
3540011     return ret;
3540012 }
```

os16: «lib/sys/stat.h»

«

Si veda la sezione u0.2.

```
3550001 #ifndef _SYS_STAT_H
3550002 #define _SYS_STAT_H     1
3550003
3550004 #include <restrict.h>
```

```
3550005 #include <const.h>
3550006 #include <sys/types.h>  // dev_t
3550007                        // off_t
3550008                        // blkcnt_t
3550009                        // blksize_t
3550010                        // ino_t
3550011                        // mode_t
3550012                        // nlink_t
3550013                        // uid_t
3550014                        // gid_t
3550015                        // time_t
3550016 //------------------------------------------------------------------------
3550017 // File type.
3550018 //------------------------------------------------------------------------
3550019 #define S_IFMT   0170000        // File type mask.
3550020 //
3550021 #define S_IFBLK  0060000        // Block device file.
3550022 #define S_IFCHR  0020000        // Character device file.
3550023 #define S_IFIFO  0010000        // Pipe (FIFO) file.
3550024 #define S_IFREG  0100000        // Regular file.
3550025 #define S_IFDIR  0040000        // Directory.
3550026 #define S_IFLNK  0120000        // Symbolic link.
3550027 #define S_IFSOCK 0140000        // Unix domain socket.
3550028 //------------------------------------------------------------------------
3550029 // Owner user access permissions.
3550030 //------------------------------------------------------------------------
3550031 #define S_IRWXU  0000700        // Owner user access permissions mask.
3550032 //
3550033 #define S_IRUSR  0000400        // Owner user read access permission.
3550034 #define S_IWUSR  0000200        // Owner user write access permission.
3550035 #define S_IXUSR  0000100        // Owner user execution or cross perm.
3550036 //------------------------------------------------------------------------
3550037 // Group owner access permissions.
3550038 //------------------------------------------------------------------------
3550039 #define S_IRWXG  0000070        // Owner group access permissions mask.
3550040 //
3550041 #define S_IRGRP  0000040        // Owner group read access permission.
3550042 #define S_IWGRP  0000020        // Owner group write access permission.
3550043 #define S_IXGRP  0000010        // Owner group execution or cross perm.
3550044 //------------------------------------------------------------------------
3550045 // Other users access permissions.
3550046 //------------------------------------------------------------------------
3550047 #define S_IRWXO  0000007        // Other users access permissions mask.
3550048 //
3550049 #define S_IROTH  0000004        // Other users read access permission.
3550050 #define S_IWOTH  0000002        // Other users write access permissions.
3550051 #define S_IXOTH  0000001        // Other users execution or cross perm.
3550052 //------------------------------------------------------------------------
3550053 // S-bit: in this case there is no mask to select all of them.
3550054 //------------------------------------------------------------------------
3550055 #define S_ISUID  0004000        // S-UID.
3550056 #define S_ISGID  0002000        // S-GID.
3550057 #define S_ISVTX  0001000        // Sticky.
3550058 //------------------------------------------------------------------------
3550059 // Macro-instructions to verify the type of file.
3550060 //------------------------------------------------------------------------
3550061 #define S_ISBLK(m)   (((m) & S_IFMT) == S_IFBLK)   // Block device.
3550062 #define S_ISCHR(m)   (((m) & S_IFMT) == S_IFCHR)   // Character device.
3550063 #define S_ISFIFO(m)  (((m) & S_IFMT) == S_IFIFO)   // FIFO.
3550064 #define S_ISREG(m)   (((m) & S_IFMT) == S_IFREG)   // Regular file.
3550065 #define S_ISDIR(m)   (((m) & S_IFMT) == S_IFDIR)   // Directory.
3550066 #define S_ISLNK(m)   (((m) & S_IFMT) == S_IFLNK)   // Symbolic link.
3550067 #define S_ISSOCK(m)  (((m) & S_IFMT) == S_IFSOCK)  // Socket.
3550068 //------------------------------------------------------------------------
3550069 // Structure 'stat'.
3550070 //------------------------------------------------------------------------
3550071 struct stat {
3550072     dev_t     st_dev;       // Device containing the file.
3550073     ino_t     st_ino;       // File serial number (inode number).
3550074     mode_t    st_mode;      // File type and permissions.
3550075     nlink_t   st_nlink;     // Links to the file.
3550076     uid_t     st_uid;       // Owner user id.
3550077     gid_t     st_gid;       // Owner group id.
3550078     dev_t     st_rdev;      // Device number if it is a device file.
3550079     off_t     st_size;      // File size.
3550080     time_t    st_atime;     // Last access time.
3550081     time_t    st_mtime;     // Last modification time.
3550082     time_t    st_ctime;     // Last inode modification.
3550083     blksize_t st_blksize;   // Block size for I/O operations.
3550084     blkcnt_t  st_blocks;    // File size / block size.
3550085 };
3550086 //------------------------------------------------------------------------
3550087 // Function prototypes.
3550088 //------------------------------------------------------------------------
3550089 int    chmod  (const char *path, mode_t mode);
3550090 int    fchmod (int fdn, mode_t mode);
3550091 int    fstat  (int fdn, struct stat *buffer);
3550092 int    lstat  (const char *restrict path, struct stat *restrict buffer);
3550093 int    mkdir  (const char *path, mode_t mode);
3550094 int    mkfifo (const char *path, mode_t mode);
3550095 int    mknod  (const char *path, mode_t mode, dev_t dev);
3550096 int    stat   (const char *restrict path, struct stat *restrict buffer);
3550097 mode_t umask  (mode_t mask);
3550098
3550099 #endif // _SYS_STAT_H
```


lib/sys/stat/chmod.c

«

Si veda la sezione u0.4.

```
3560001 #include <sys/stat.h>
3560002 #include <string.h>
3560003 #include <const.h>
3560004 #include <sys/os16.h>
3560005 #include <errno.h>
3560006 #include <limits.h>
3560007 //------------------------------------------------------------------------
3560008 int
3560009 chmod (const char *path, mode_t mode)
3560010 {
3560011     sysmsg_chmod_t msg;
3560012     //
3560013     strncpy (msg.path, path, PATH_MAX);
3560014     msg.mode = mode;
3560015     //
3560016     sys (SYS_CHMOD, &msg, (sizeof msg));
3560017     //
3560018     errno = msg.errno;
3560019     errln = msg.errln;
3560020     strncpy (errfn, msg.errfn, PATH_MAX);
3560021     return (msg.ret);
3560022 }
```

lib/sys/stat/fchmod.c

«

Si veda la sezione u0.4.

```
3570001 #include <sys/stat.h>
3570002 #include <string.h>
3570003 #include <const.h>
3570004 #include <sys/os16.h>
3570005 #include <errno.h>
3570006 #include <limits.h>
3570007 //------------------------------------------------------------------------
3570008 int
3570009 fchmod (int fdn, mode_t mode)
3570010 {
3570011     sysmsg_fchmod_t msg;
3570012     //
3570013     msg.fdn  = fdn;
3570014     msg.mode = mode;
3570015     //
3570016     sys (SYS_FCHMOD, &msg, (sizeof msg));
3570017     //
3570018     errno = msg.errno;
3570019     errln = msg.errln;
3570020     strncpy (errfn, msg.errfn, PATH_MAX);
3570021     return (msg.ret);
3570022 }
```

lib/sys/stat/fstat.c

«

Si veda la sezione u0.36.

```
3580001 #include <unistd.h>
3580002 #include <errno.h>
3580003 #include <sys/os16.h>
3580004 #include <string.h>
3580005 //------------------------------------------------------------------------
3580006 int
3580007 fstat (int fdn, struct stat *buffer)
3580008 {
3580009     sysmsg_fstat_t msg;
3580010     //
3580011     msg.fdn             = fdn;
3580012     msg.stat.st_dev     = buffer->st_dev;
3580013     msg.stat.st_ino     = buffer->st_ino;
3580014     msg.stat.st_mode    = buffer->st_mode;
3580015     msg.stat.st_nlink   = buffer->st_nlink;
3580016     msg.stat.st_uid     = buffer->st_uid;
3580017     msg.stat.st_gid     = buffer->st_gid;
3580018     msg.stat.st_rdev    = buffer->st_rdev;
3580019     msg.stat.st_size    = buffer->st_size;
3580020     msg.stat.st_atime   = buffer->st_atime;
3580021     msg.stat.st_mtime   = buffer->st_mtime;
3580022     msg.stat.st_ctime   = buffer->st_ctime;
3580023     msg.stat.st_blksize = buffer->st_blksize;
3580024     msg.stat.st_blocks  = buffer->st_blocks;
3580025     //
3580026     sys (SYS_FSTAT, &msg, (sizeof msg));
3580027     //
3580028     buffer->st_dev      = msg.stat.st_dev;
3580029     buffer->st_ino      = msg.stat.st_ino;
3580030     buffer->st_mode     = msg.stat.st_mode;
3580031     buffer->st_nlink    = msg.stat.st_nlink;
3580032     buffer->st_uid      = msg.stat.st_uid;
3580033     buffer->st_gid      = msg.stat.st_gid;
3580034     buffer->st_rdev     = msg.stat.st_rdev;
3580035     buffer->st_size     = msg.stat.st_size;
3580036     buffer->st_atime    = msg.stat.st_atime;
3580037     buffer->st_mtime    = msg.stat.st_mtime;
3580038     buffer->st_ctime    = msg.stat.st_ctime;
3580039     buffer->st_blksize  = msg.stat.st_blksize;
3580040     buffer->st_blocks   = msg.stat.st_blocks;
3580041     //
3580042     errno = msg.errno;
```

```
3580043     errln = msg.errln;
3580044     strncpy (errfn, msg.errfn, PATH_MAX);
3580045     return (msg.ret);
3580046 }
```

lib/sys/stat/mkdir.c «

Si veda la sezione u0.25.

```
3590001 #include <sys/stat.h>
3590002 #include <string.h>
3590003 #include <const.h>
3590004 #include <sys/os16.h>
3590005 #include <errno.h>
3590006 #include <limits.h>
3590007 //------------------------------------------------------------------------
3590008 int
3590009 mkdir (const char *path, mode_t mode)
3590010 {
3590011     sysmsg_mkdir_t msg;
3590012     //
3590013     strncpy (msg.path, path, PATH_MAX);
3590014     msg.mode = mode;
3590015     //
3590016     sys (SYS_MKDIR, &msg, (sizeof msg));
3590017     //
3590018     errno = msg.errno;
3590019     errln = msg.errln;
3590020     strncpy (errfn, msg.errfn, PATH_MAX);
3590021     return (msg.ret);
3590022 }
```

lib/sys/stat/mknod.c «

Si veda la sezione u0.26.

```
3600001 #include <unistd.h>
3600002 #include <errno.h>
3600003 #include <sys/os16.h>
3600004 #include <string.h>
3600005 //------------------------------------------------------------------------
3600006 int
3600007 mknod (const char *path, mode_t mode, dev_t device)
3600008 {
3600009     sysmsg_mknod_t msg;
3600010     //
3600011     strncpy (msg.path, path, PATH_MAX);
3600012     msg.mode   = mode;
3600013     msg.device = device;
3600014     //
3600015     sys (SYS_MKNOD, &msg, (sizeof msg));
3600016     //
3600017     errno = msg.errno;
3600018     errln = msg.errln;
3600019     strncpy (errfn, msg.errfn, PATH_MAX);
3600020     return (msg.ret);
3600021 }
```

lib/sys/stat/stat.c «

Si veda la sezione u0.36.

```
3610001 #include <unistd.h>
3610002 #include <errno.h>
3610003 #include <sys/os16.h>
3610004 #include <string.h>
3610005 //------------------------------------------------------------------------
3610006 int
3610007 stat (const char *path, struct stat *buffer)
3610008 {
3610009     sysmsg_stat_t msg;
3610010     //
3610011     strncpy (msg.path, path, PATH_MAX);
3610012     //
3610013     msg.stat.st_dev     = buffer->st_dev;
3610014     msg.stat.st_ino     = buffer->st_ino;
3610015     msg.stat.st_mode    = buffer->st_mode;
3610016     msg.stat.st_nlink   = buffer->st_nlink;
3610017     msg.stat.st_uid     = buffer->st_uid;
3610018     msg.stat.st_gid     = buffer->st_gid;
3610019     msg.stat.st_rdev    = buffer->st_rdev;
3610020     msg.stat.st_size    = buffer->st_size;
3610021     msg.stat.st_atime   = buffer->st_atime;
3610022     msg.stat.st_mtime   = buffer->st_mtime;
3610023     msg.stat.st_ctime   = buffer->st_ctime;
3610024     msg.stat.st_blksize = buffer->st_blksize;
3610025     msg.stat.st_blocks  = buffer->st_blocks;
3610026     //
3610027     sys (SYS_STAT, &msg, (sizeof msg));
3610028     //
3610029     buffer->st_dev      = msg.stat.st_dev;
3610030     buffer->st_ino      = msg.stat.st_ino;
3610031     buffer->st_mode     = msg.stat.st_mode;
3610032     buffer->st_nlink    = msg.stat.st_nlink;
3610033     buffer->st_uid      = msg.stat.st_uid;
3610034     buffer->st_gid      = msg.stat.st_gid;
3610035     buffer->st_rdev     = msg.stat.st_rdev;
3610036     buffer->st_size     = msg.stat.st_size;
```



```
3610037     buffer->st_atime    = msg.stat.st_atime;
3610038     buffer->st_mtime    = msg.stat.st_mtime;
3610039     buffer->st_ctime    = msg.stat.st_ctime;
3610040     buffer->st_blksize  = msg.stat.st_blksize;
3610041     buffer->st_blocks   = msg.stat.st_blocks;
3610042     //
3610043     errno = msg.errno;
3610044     errln = msg.errln;
3610045     strncpy (errfn, msg.errfn, PATH_MAX);
3610046     return (msg.ret);
3610047 }
```

lib/sys/stat/umask.c «

Si veda la sezione u0.40.

```
3620001 #include <sys/stat.h>
3620002 #include <string.h>
3620003 #include <const.h>
3620004 #include <sys/os16.h>
3620005 #include <errno.h>
3620006 #include <limits.h>
3620007 //------------------------------------------------------------------------
3620008 mode_t
3620009 umask (mode_t mask)
3620010 {
3620011     sysmsg_umask_t msg;
3620012     msg.umask = mask;
3620013     sys (SYS_UMASK, &msg, (sizeof msg));
3620014     return (msg.ret);
3620015 }
```

os16: «lib/sys/types.h»

Si veda la sezione u0.2. «

```
3630001 #ifndef _SYS_TYPES_H
3630002 #define _SYS_TYPES_H    1
3630003 //------------------------------------------------------------------------
3630004
3630005 #include <clock_t.h>
3630006 #include <time_t.h>
3630007 #include <size_t.h>
3630008 #include <stdint.h>
3630009 //------------------------------------------------------------------------
3630010 typedef     long int blkcnt_t;
3630011 typedef     long int blksize_t;
3630012 typedef     uint16_t dev_t;     // Traditional device size.
3630013 typedef unsigned int id_t;
3630014 typedef unsigned int gid_t;
3630015 typedef unsigned int uid_t;
3630016 typedef     uint16_t ino_t;     // Minix 1 file system inode size.
3630017 typedef     uint16_t mode_t;    // Minix 1 file system mode size.
3630018 typedef unsigned int nlink_t;
3630019 typedef     long int off_t;
3630020 typedef          int pid_t;
3630021 typedef unsigned int pthread_t;
3630022 typedef     long int ssize_t;
3630023 //------------------------------------------------------------------------
3630024 // Common extentions.
3630025 //
3630026 dev_t makedev (int major, int minor);
3630027 int   major   (dev_t device);
3630028 int   minor   (dev_t device);
3630029 //------------------------------------------------------------------------
3630030
3630031 #endif
```

lib/sys/types/major.c «

Si veda la sezione u0.65.

```
3640001 #include <sys/types.h>
3640002 //------------------------------------------------------------------------
3640003 int
3640004 major (dev_t device)
3640005 {
3640006     return ((int) (device / 256));
3640007 }
```

lib/sys/types/makedev.c «

Si veda la sezione u0.65.

```
3650001 #include <sys/types.h>
3650002 //------------------------------------------------------------------------
3650003 dev_t
3650004 makedev (int major, int minor)
3650005 {
3650006     return ((dev_t) (major * 256 + minor));
3650007 }
```

lib/sys/types/minor.c

«

Si veda la sezione u0.65.

```
3660001 #include <sys/types.h>
3660002 //-------------------------------------------------------------------------
3660003 int
3660004 minor (dev_t device)
3660005 {
3660006     return ((dev_t) (device & 0x00FF));
3660007 }
```

os16: «lib/sys/wait.h»

«

Si veda la sezione u0.2.

```
3670001 #ifndef _SYS_WAIT_H
3670002 #define _SYS_WAIT_H      1
3670003
3670004 #include <sys/types.h>
3670005
3670006 //-------------------------------------------------------------------------
3670007 pid_t wait    (int *status);
3670008 //-------------------------------------------------------------------------
3670009
3670010 #endif
```

lib/sys/wait/wait.c

«

Si veda la sezione u0.43.

```
3680001 #include <sys/types.h>
3680002 #include <errno.h>
3680003 #include <sys/os16.h>
3680004 #include <stddef.h>
3680005 #include <string.h>
3680006 //-------------------------------------------------------------------------
3680007 pid_t
3680008 wait (int *status)
3680009 {
3680010     sysmsg_wait_t msg;
3680011     msg.ret    = 0;
3680012     msg.errno  = 0;
3680013     msg.status = 0;
3680014     while (msg.ret == 0)
3680015       {
3680016         //
3680017         // Loop as long as there are children, an none is dead.
3680018         //
3680019         sys (SYS_WAIT, &msg, (sizeof msg));
3680020       }
3680021     errno = msg.errno;
3680022     errln = msg.errln;
3680023     strncpy (errfn, msg.errfn, PATH_MAX);
3680024     //
3680025     if (status != NULL)
3680026       {
3680027         //
3680028         // Only the low eight bits are returned.
3680029         //
3680030         *status = (msg.status & 0x00FF);
3680031       }
3680032     return (msg.ret);
3680033 }
```

os16: «lib/time.h»

«

Si veda la sezione u0.2.

```
3690001 #ifndef _TIME_H
3690002 #define _TIME_H       1
3690003 //-------------------------------------------------------------------------
3690004
3690005 #include <const.h>
3690006 #include <restrict.h>
3690007 #include <size_t.h>
3690008 #include <time_t.h>
3690009 #include <clock_t.h>
3690010 #include <NULL.h>
3690011 #include <stdint.h>
3690012 //-------------------------------------------------------------------------
3690013 #define CLOCKS_PER_SEC  18  // Should be 18.22 Hz, but it is a 'int'.
3690014 //-------------------------------------------------------------------------
3690015 struct tm {int tm_sec;  int tm_min;  int tm_hour;
3690016            int tm_mday; int tm_mon;  int tm_year;
3690017            int tm_wday; int tm_yday; int tm_isdst;};
3690018 //-------------------------------------------------------------------------
3690019 clock_t    clock     (void);
3690020 time_t     time      (time_t *timer);
3690021 int        stime     (time_t *timer);
3690022 double     difftime  (time_t time1, time_t time0);
3690023 time_t     mktime    (struct tm *timeptr);
3690024 struct tm *gmtime    (const time_t *timer);
3690025 struct tm *localtime (const time_t *timer);
3690026 char      *asctime   (const struct tm *timeptr);
3690027 char      *ctime     (const time_t *timer);
3690028 size_t     strftime  (char * restrict s, size_t maxsize,
3690029                       const char * restrict format,
3690030                       const struct tm * restrict timeptr);
```



```
3690031 //-------------------------------------------------------------------------
3690032 #define difftime(t1,t0) ((double)((t1)-(t0)))
3690033 #define ctime(t)        (asctime (localtime (t)))
3690034 #define localtime(t)    (gmtime (t))
3690035 //-------------------------------------------------------------------------
3690036
3690037 #endif
```

lib/time/asctime.c

«

Si veda la sezione u0.13.

```
3700001 #include <time.h>
3700002 #include <string.h>
3700003 #include <stdio.h>
3700004
3700005 //-------------------------------------------------------------------------
3700006 char *
3700007 asctime (const struct tm *timeptr)
3700008 {
3700009     static char time_string[25]; // 'Sun Jan 30 24:00:00 2111'
3700010     //
3700011     // Check argument.
3700012     //
3700013     if (timeptr == NULL)
3700014       {
3700015         return (NULL);
3700016       }
3700017     //
3700018     // Set week day.
3700019     //
3700020     switch (timeptr->tm_wday)
3700021       {
3700022         case 0:
3700023             strcpy (&time_string[0], "Sun");
3700024             break;
3700025         case 1:
3700026             strcpy (&time_string[0], "Mon");
3700027             break;
3700028         case 2:
3700029             strcpy (&time_string[0], "Tue");
3700030             break;
3700031         case 3:
3700032             strcpy (&time_string[0], "Wed");
3700033             break;
3700034         case 4:
3700035             strcpy (&time_string[0], "Thu");
3700036             break;
3700037         case 5:
3700038             strcpy (&time_string[0], "Fri");
3700039             break;
3700040         case 6:
3700041             strcpy (&time_string[0], "Sat");
3700042             break;
3700043         default:
3700044             strcpy (&time_string[0], "Err");
3700045       }
3700046     //
3700047     // Set month.
3700048     //
3700049     switch (timeptr->tm_mon)
3700050       {
3700051         case 1:
3700052             strcpy (&time_string[3], " Jan");
3700053             break;
3700054         case 2:
3700055             strcpy (&time_string[3], " Feb");
3700056             break;
3700057         case 3:
3700058             strcpy (&time_string[3], " Mar");
3700059             break;
3700060         case 4:
3700061             strcpy (&time_string[3], " Apr");
3700062             break;
3700063         case 5:
3700064             strcpy (&time_string[3], " May");
3700065             break;
3700066         case 6:
3700067             strcpy (&time_string[3], " Jun");
3700068             break;
3700069         case 7:
3700070             strcpy (&time_string[3], " Jul");
3700071             break;
3700072         case 8:
3700073             strcpy (&time_string[3], " Aug");
3700074             break;
3700075         case 9:
3700076             strcpy (&time_string[3], " Sep");
3700077             break;
3700078         case 10:
3700079             strcpy (&time_string[3], " Oct");
3700080             break;
3700081         case 11:
3700082             strcpy (&time_string[3], " Nov");
3700083             break;
3700084         case 12:
3700085             strcpy (&time_string[3], " Dec");
3700086             break;
3700087         default:
3700088             strcpy (&time_string[3], " Err");
3700089       }
```

```
3700090     //
3700091     // Set day of month, hour, minute, second and year.
3700092     //
3700093     sprintf (&time_string[7], " %2i %2i:%2i:%2i %4i",
3700094              timeptr->tm_mday,  timeptr->tm_hour,  timeptr->tm_min,
3700095              timeptr->tm_sec,  timeptr->tm_year);
3700096     //
3700097     //
3700098     //
3700099     return (&time_string[0]);
3700100 }
```

lib/time/clock.c

«

Si veda la sezione u0.6.

```
3710001 #include <time.h>
3710002 #include <sys/os16.h>
3710003 //---------------------------------------------------------------------------
3710004 clock_t
3710005 clock (void)
3710006 {
3710007     sysmsg_clock_t msg;
3710008     msg.ret = 0;
3710009     sys (SYS_CLOCK, &msg, (sizeof msg));
3710010     return (msg.ret);
3710011 }
3710012
```

lib/time/gmtime.c

«

Si veda la sezione u0.13.

```
3720001 #include <time.h>
3720002 //---------------------------------------------------------------------------
3720003 static int leap_year (int year);
3720004 //---------------------------------------------------------------------------
3720005 struct tm *
3720006 gmtime (const time_t *timer)
3720007 {
3720008     static struct tm  tms;
3720009     int               loop;
3720010     unsigned int      remainder;
3720011     unsigned int      days;
3720012     //
3720013     // Check argument.
3720014     //
3720015     if (timer == NULL)
3720016       {
3720017         return (NULL);
3720018       }
3720019     //
3720020     // Days since epoch. There are 86400 seconds per day.
3720021     // At the moment, the field 'tm_yday' will contain
3720022     // all days since epoch.
3720023     //
3720024     days      = *timer / 86400L;
3720025     remainder = *timer % 86400L;
3720026     //
3720027     // Minutes, after full days.
3720028     //
3720029     tms.tm_min = remainder / 60U;
3720030     //
3720031     // Seconds, after full minutes.
3720032     //
3720033     tms.tm_sec = remainder % 60U;
3720034     //
3720035     // Hours, after full days.
3720036     //
3720037     tms.tm_hour = tms.tm_min / 60;
3720038     //
3720039     // Minutes, after full hours.
3720040     //
3720041     tms.tm_min = tms.tm_min % 60;
3720042     //
3720043     // Find the week day. Must remove some days to align the
3720044     // calculation. So: the week days of the first week of 1970
3720045     // are not valid! After 1970-01-04 calculations are right.
3720046     //
3720047     tms.tm_wday = (days - 3) % 7;
3720048     //
3720049     // Find the year: the field 'tm_yday' will be reduced to the days
3720050     // of current year.
3720051     //
3720052     for (tms.tm_year = 1970; days > 0; tms.tm_year++)
3720053       {
3720054         if (leap_year (tms.tm_year))
3720055           {
3720056             if (days >= 366)
3720057               {
3720058                 days -= 366;
3720059                 continue;
3720060               }
3720061             else
3720062               {
3720063                 break;
3720064               }
3720065           }
3720066         else
3720067           {
```



```
3720068             if (days >= 365)
3720069               {
3720070                 days -= 365;
3720071                 continue;
3720072               }
3720073             else
3720074               {
3720075                 break;
3720076               }
3720077           }
3720078       }
3720079     //
3720080     // Day of the year.
3720081     //
3720082     tms.tm_yday = days + 1;
3720083     //
3720084     // Find the month.
3720085     //
3720086     tms.tm_mday = days + 1;
3720087     //
3720088     for (tms.tm_mon = 0, loop = 1; tms.tm_mon <= 12 && loop;)
3720089       {
3720090         tms.tm_mon++;
3720091         //
3720092         switch (tms.tm_mon)
3720093           {
3720094             case 1:
3720095             case 3:
3720096             case 5:
3720097             case 7:
3720098             case 8:
3720099             case 10:
3720100             case 12:
3720101                 if (tms.tm_mday >= 31)
3720102                   {
3720103                     tms.tm_mday -= 31;
3720104                   }
3720105                 else
3720106                   {
3720107                     loop = 0;
3720108                   }
3720109                 break;
3720110             case 4:
3720111             case 6:
3720112             case 9:
3720113             case 11:
3720114                 if (tms.tm_mday >= 30)
3720115                   {
3720116                     tms.tm_mday -= 30;
3720117                   }
3720118                 else
3720119                   {
3720120                     loop = 0;
3720121                   }
3720122                 break;
3720123             case 2:
3720124                 if (leap_year (tms.tm_year))
3720125                   {
3720126                     if (tms.tm_mday >= 29)
3720127                       {
3720128                         tms.tm_mday -= 29;
3720129                       }
3720130                     else
3720131                       {
3720132                         loop = 0;
3720133                       }
3720134                   }
3720135                 else
3720136                   {
3720137                     if (tms.tm_mday >= 28)
3720138                       {
3720139                         tms.tm_mday -= 28;
3720140                       }
3720141                     else
3720142                       {
3720143                         loop = 0;
3720144                       }
3720145                   }
3720146                 break;
3720147           }
3720148       }
3720149     //
3720150     // No check for day light saving time.
3720151     //
3720152     tms.tm_isdst = 0;
3720153     //
3720154     // Return.
3720155     //
3720156     return (&tms);
3720157 }
3720158 //---------------------------------------------------------------------------
3720159 static int
3720160 leap_year (int year)
3720161 {
3720162     if ((year % 4) == 0)
3720163       {
3720164         if ((year % 100) == 0)
3720165           {
3720166             if ((year % 400) == 0)
3720167               {
3720168                 return (1);
```

```
3720169               }
3720170             else
3720171               {
3720172                 return (0);
3720173               }
3720174           }
3720175         else
3720176           {
3720177             return (1);
3720178           }
3720179       }
3720180     else
3720181       {
3720182         return (0);
3720183       }
3720184 }
```

lib/time/mktime.c

«

Si veda la sezione u0.13.

```
3730001 #include <time.h>
3730002 #include <string.h>
3730003 #include <stdio.h>
3730004 //------------------------------------------------------------------------
3730005 static int leap_year (int year);
3730006 //------------------------------------------------------------------------
3730007 time_t
3730008 mktime (const struct tm *timeptr)
3730009 {
3730010     time_t timer_total;
3730011     time_t timer_aux;
3730012     int    days;
3730013     int    month;
3730014     int    year;
3730015     //
3730016     // From seconds to days.
3730017     //
3730018     timer_total  = timeptr->tm_sec;
3730019     //
3730020     timer_aux    = timeptr->tm_min;
3730021     timer_aux   *= 60;
3730022     timer_total += timer_aux;
3730023     //
3730024     timer_aux    = timeptr->tm_hour;
3730025     timer_aux   *= (60 * 60);
3730026     timer_total += timer_aux;
3730027     //
3730028     timer_aux    = timeptr->tm_mday;
3730029     timer_aux   *= 24;
3730030     timer_aux   *= (60 * 60);
3730031     timer_total += timer_aux;
3730032     //
3730033     // Month: add the days of months.
3730034     // Will scan the months, from the first, but before the
3730035     // months of the value inside field 'tm_mon'.
3730036     //
3730037     for (month = 1, days = 0; month < timeptr->tm_mon; month++)
3730038       {
3730039         switch (month)
3730040           {
3730041             case 1:
3730042             case 3:
3730043             case 5:
3730044             case 7:
3730045             case 8:
3730046             case 10:
3730047                 //
3730048                 // There is no December, because the scan can go up to
3730049                 // the month before the value inside field 'tm_mon'.
3730050                 //
3730051                 days += 31;
3730052                 break;
3730053             case 4:
3730054             case 6:
3730055             case 9:
3730056             case 11:
3730057                 days += 30;
3730058                 break;
3730059             case 2:
3730060                 if (leap_year (timeptr->tm_year))
3730061                   {
3730062                     days += 29;
3730063                   }
3730064                 else
3730065                   {
3730066                     days += 28;
3730067                   }
3730068                 break;
3730069           }
3730070       }
3730071     //
3730072     timer_aux    = days;
3730073     timer_aux   *= 24;
3730074     timer_aux   *= (60 * 60);
3730075     timer_total += timer_aux;
3730076     //
3730077     // Year. The work is similar to the one of months: days of
3730078     // years are counted, up to the year before the one reported
3730079     // by the field 'tm_year'.
3730080     //
```



```
3730081     for (year = 1970, days = 0; year < timeptr->tm_year; year++)
3730082       {
3730083         if (leap_year (year))
3730084           {
3730085             days += 366;
3730086           }
3730087         else
3730088           {
3730089             days += 365;
3730090           }
3730091       }
3730092     //
3730093     // After all, must subtract a day from the total.
3730094     //
3730095     days--;
3730096     //
3730097     timer_aux    = days;
3730098     timer_aux   *= 24;
3730099     timer_aux   *= (60 * 60);
3730100     timer_total += timer_aux;
3730101     //
3730102     // That's all.
3730103     //
3730104     return (timer_total);
3730105 }
3730106 //------------------------------------------------------------------------
3730107 int
3730108 leap_year (int year)
3730109 {
3730110     if ((year % 4) == 0)
3730111       {
3730112         if ((year % 100) == 0)
3730113           {
3730114             if ((year % 400) == 0)
3730115               {
3730116                 return (1);
3730117               }
3730118             else
3730119               {
3730120                 return (0);
3730121               }
3730122           }
3730123         else
3730124           {
3730125             return (1);
3730126           }
3730127       }
3730128     else
3730129       {
3730130         return (0);
3730131       }
3730132 }
```

lib/time/stime.c

«

Si veda la sezione u0.39.

```
3740001 #include <time.h>
3740002 #include <sys/os16.h>
3740003 //------------------------------------------------------------------------
3740004 int
3740005 stime (time_t *timer)
3740006 {
3740007     sysmsg_stime_t msg;
3740008     msg.timer = *timer;
3740009     msg.ret = 0;
3740010     sys (SYS_STIME, &msg, (sizeof msg));
3740011     return (msg.ret);
3740012 }
```

lib/time/time.c

«

Si veda la sezione u0.39.

```
3750001 #include <time.h>
3750002 #include <sys/os16.h>
3750003 //------------------------------------------------------------------------
3750004 time_t
3750005 time (time_t *timer)
3750006 {
3750007     sysmsg_time_t msg;
3750008     msg.ret = ((time_t) 0);
3750009     sys (SYS_TIME, &msg, (sizeof msg));
3750010     if (timer != NULL)
3750011       {
3750012         *timer = msg.ret;
3750013       }
3750014     return (msg.ret);
3750015 }
```

os16: «lib/unistd.h»

«

Si veda la sezione u0.2.

```
3760001 #ifndef _UNISTD_H
3760002 #define _UNISTD_H       1
3760003
3760004 #include <const.h>
```

```
3760005 #include <sys/stat.h>
3760006 #include <sys/os16.h>
3760007 #include <sys/types.h>  // size_t, ssize_t, uid_t, gid_t, off_t, pid_t
3760008 #include <inttypes.h>   // intptr_t
3760009 #include <SEEK.h>       // SEEK_CUR, SEEK_SET, SEEK_END
3760010 //------------------------------------------------------------------------
3760011 extern char **environ;  // Variable 'environ' is used by functions like
3760012                         // 'execv()' in replacement for 'envp[][]'.
3760013 //------------------------------------------------------------------------
3760014 extern char *optarg;    // Used by 'optarg()'.
3760015 extern int   optind;    //
3760016 extern int   opterr;    //
3760017 extern int   optopt;    //
3760018 //------------------------------------------------------------------------
3760019 #define STDIN_FILENO    0       //
3760020 #define STDOUT_FILENO   1       // Standard file descriptors.
3760021 #define STDERR_FILENO   2       //
3760022 //------------------------------------------------------------------------
3760023 #define R_OK            4       // Read permission.
3760024 #define W_OK            2       // Write permission.
3760025 #define X_OK            1       // Execute or traverse permission.
3760026 #define F_OK            0       // File exists.
3760027 //------------------------------------------------------------------------
3760028
3760029 int          access      (const char *path, int mode);
3760030 int          chdir       (const char *path);
3760031 int          chown       (const char *path, uid_t uid, gid_t gid);
3760032 int          close       (int fdn);
3760033 int          dup         (int fdn_old);
3760034 int          dup2        (int fdn_old, int fdn_new);
3760035 int          execl       (const char *path, const char *arg, ...);
3760036 int          execle      (const char *path, const char *arg, ...);
3760037 int          execlp      (const char *path, const char *arg, ...);
3760038 int          execv       (const char *path, char *const argv[]);
3760039 int          execve      (const char *path, char *const argv[],
3760040                           char *const envp[]);
3760041 int          execvp      (const char *path, char *const argv[]);
3760042 void        _exit        (int status);
3760043 int          fchown      (int fdn, uid_t uid, gid_t gid);
3760044 pid_t        fork        (void);
3760045 char        *getcwd      (char *buffer, size_t size);
3760046 uid_t        geteuid     (void);
3760047 int          getopt      (int argc, char *const argv[],
3760048                           const char *optstring);
3760049 pid_t        getpgrp     (void);
3760050 pid_t        getppid     (void);
3760051 pid_t        getpid      (void);
3760052 uid_t        getuid      (void);
3760053 int          isatty      (int fdn);
3760054 int          link        (const char *path_old, const char *path_new);
3760055 off_t        lseek       (int fdn, off_t offset, int whence);
3760056 #define      nice(n)     (0)
3760057 ssize_t      read        (int fdn, void *buffer, size_t count);
3760058 #define      readlink(p,b,s) ((ssize_t) -1)
3760059 int          rmdir       (const char *path);
3760060 int          seteuid     (uid_t uid);
3760061 int          setpgrp     (void);
3760062 int          setuid      (uid_t uid);
3760063 unsigned int sleep       (unsigned int s);
3760064 #define      sync()      /**/
3760065 char        *ttyname     (int fdn);
3760066 int          unlink      (const char *path);
3760067 ssize_t      write       (int fdn, const void *buffer, size_t count);
3760068
3760069 #endif
```

lib/unistd/_exit.c

«

Si veda la sezione u0.2.

```
3770001 #include <unistd.h>
3770002 #include <sys/os16.h>
3770003 //------------------------------------------------------------------------
3770004 void
3770005 _exit (int status)
3770006 {
3770007     sysmsg_exit_t msg;
3770008     //
3770009     // Only the low eight bit are returned.
3770010     //
3770011     msg.status = (status & 0xFF);
3770012     //
3770013     //
3770014     //
3770015     sys (SYS_EXIT, &msg, (sizeof msg));
3770016     //
3770017     // Should not return from system call, but if it does, loop
3770018     // forever:
3770019     //
3770020     while (1);
3770021 }
```

lib/unistd/access.c

«

Si veda la sezione u0.1.

```
3780001 #include <unistd.h>
3780002 #include <sys/stat.h>
3780003 #include <errno.h>
3780004 //------------------------------------------------------------------------
```



```
3780005 int
3780006 access (const char *path, int mode)
3780007 {
3780008     struct stat st;
3780009     int         status;
3780010     uid_t       euid;
3780011     //
3780012     status = stat (path, &st);
3780013     if (status != 0)
3780014       {
3780015         return (-1);
3780016       }
3780017     //
3780018     // File exists?
3780019     //
3780020     if (mode == F_OK)
3780021       {
3780022         return (0);
3780023       }
3780024     //
3780025     // Some access permissions are requested: get effective user id.
3780026     //
3780027     euid = geteuid ();
3780028     //
3780029     // Check owner access permissions.
3780030     //
3780031     if (st.st_uid == euid && ((st.st_mode & S_IRWXU) == (mode << 6)))
3780032       {
3780033         return (0);
3780034       }
3780035     //
3780036     // Check others access permissions.
3780037     //
3780038     if ((st.st_mode & S_IRWXO) == (mode))
3780039       {
3780040         return (0);
3780041       }
3780042     //
3780043     // Otherwise there are no access permissions.
3780044     //
3780045     errset (EACCES);                    // Permission denied.
3780046     return (-1);
3780047 }
```

lib/unistd/chdir.c

«

Si veda la sezione u0.3.

```
3790001 #include <unistd.h>
3790002 #include <string.h>
3790003 #include <const.h>
3790004 #include <sys/os16.h>
3790005 #include <errno.h>
3790006 #include <limits.h>
3790007 //------------------------------------------------------------------------
3790008 int
3790009 chdir (const char *path)
3790010 {
3790011     sysmsg_chdir_t msg;
3790012     //
3790013     msg.ret    = 0;
3790014     msg.errno  = 0;
3790015     //
3790016     strncpy (msg.path, path, PATH_MAX);
3790017     //
3790018     sys (SYS_CHDIR, &msg, (sizeof msg));
3790019     //
3790020     errno = msg.errno;
3790021     errln = msg.errln;
3790022     strncpy (errfn, msg.errfn, PATH_MAX);
3790023     return (msg.ret);
3790024 }
```

lib/unistd/chown.c

«

Si veda la sezione u0.5.

```
3800001 #include <unistd.h>
3800002 #include <string.h>
3800003 #include <const.h>
3800004 #include <sys/os16.h>
3800005 #include <errno.h>
3800006 #include <limits.h>
3800007 //------------------------------------------------------------------------
3800008 int
3800009 chown (const char *path, uid_t uid, gid_t gid)
3800010 {
3800011     sysmsg_chown_t msg;
3800012     //
3800013     strncpy (msg.path, path, PATH_MAX);
3800014     msg.uid = uid;
3800015     msg.gid = gid;
3800016     //
3800017     sys (SYS_CHOWN, &msg, (sizeof msg));
3800018     //
3800019     errno = msg.errno;
3800020     errln = msg.errln;
3800021     strncpy (errfn, msg.errfn, PATH_MAX);
3800022     return (msg.ret);
3800023 }
```

lib/unistd/close.c

«

Si veda la sezione u0.7.

```
3810001 #include <unistd.h>
3810002 #include <errno.h>
3810003 #include <sys/os16.h>
3810004 #include <string.h>
3810005 //-------------------------------------------------------------------------
3810006 int
3810007 close (int fdn)
3810008 {
3810009     sysmsg_close_t msg;
3810010     msg.fdn = fdn;
3810011     sys (SYS_CLOSE, &msg, (sizeof msg));
3810012     errno = msg.errno;
3810013     errln = msg.errln;
3810014     strncpy (errfn, msg.errfn, PATH_MAX);
3810015     return (msg.ret);
3810016 }
```

lib/unistd/dup.c

«

Si veda la sezione u0.8.

```
3820001 #include <unistd.h>
3820002 #include <sys/os16.h>
3820003 #include <string.h>
3820004 #include <errno.h>
3820005 //-------------------------------------------------------------------------
3820006 int
3820007 dup (int fdn_old)
3820008 {
3820009     sysmsg_dup_t msg;
3820010     //
3820011     msg.fdn_old = fdn_old;
3820012     //
3820013     sys (SYS_DUP, &msg, (sizeof msg));
3820014     //
3820015     errno = msg.errno;
3820016     errln = msg.errln;
3820017     strncpy (errfn, msg.errfn, PATH_MAX);
3820018     return (msg.ret);
3820019 }
```

lib/unistd/dup2.c

«

Si veda la sezione u0.8.

```
3830001 #include <unistd.h>
3830002 #include <sys/os16.h>
3830003 #include <string.h>
3830004 #include <errno.h>
3830005 //-------------------------------------------------------------------------
3830006 int
3830007 dup2 (int fdn_old, int fdn_new)
3830008 {
3830009     sysmsg_dup2_t msg;
3830010     //
3830011     msg.fdn_old = fdn_old;
3830012     msg.fdn_new = fdn_new;
3830013     //
3830014     sys (SYS_DUP2, &msg, (sizeof msg));
3830015     //
3830016     errno = msg.errno;
3830017     errln = msg.errln;
3830018     strncpy (errfn, msg.errfn, PATH_MAX);
3830019     return (msg.ret);
3830020 }
```

lib/unistd/environ.c

«

Si veda la sezione u0.1.

```
3840001 #include <unistd.h>
3840002 //-------------------------------------------------------------------------
3840003 char **environ;
```

lib/unistd/execl.c

«

Si veda la sezione u0.20.

```
3850001 #include <unistd.h>
3850002 //-------------------------------------------------------------------------
3850003 int
3850004 execl (const char *path, const char *arg, ...)
3850005 {
3850006     int   argc;
3850007     char *arg_next;
3850008     char *argv[ARG_MAX/2];
3850009     //
3850010     va_list ap;
3850011     va_start (ap, arg);
3850012     //
3850013     arg_next = arg;
3850014     //
3850015     for (argc = 0; argc < ARG_MAX/2; argc++)
3850016       {
3850017         argv[argc] = arg_next;
3850018         if (argv[argc] == NULL)
3850019           {
3850020             break;      // End of arguments.
3850021           }
3850022         arg_next = va_arg (ap, char *);
3850023       }
3850024     //
3850025     return (execve (path, argv, environ));      // [1]
3850026 }
3850027 //
3850028 // The variable ‘environ’ is declared as ‘char **environ’ and is
3850029 // included from <unistd.h>.
3850030 //
```


lib/unistd/execle.c

«

Si veda la sezione u0.20.

```
3860001 #include <unistd.h>
3860002 //-------------------------------------------------------------------------
3860003 int
3860004 execle (const char *path, const char *arg, ...)
3860005 {
3860006     int    argc;
3860007     char  *arg_next;
3860008     char  *argv[ARG_MAX/2];
3860009     char **envp;
3860010     //
3860011     va_list ap;
3860012     va_start (ap, arg);
3860013     //
3860014     arg_next = arg;
3860015     //
3860016     for (argc = 0; argc < ARG_MAX/2; argc++)
3860017       {
3860018         argv[argc] = arg_next;
3860019         if (argv[argc] == NULL)
3860020           {
3860021             break;      // End of arguments.
3860022           }
3860023         arg_next = va_arg (ap, char *);
3860024       }
3860025     //
3860026     envp = va_arg (ap, char **);
3860027     //
3860028     return (execve (path, argv, envp));
3860029 }
```

lib/unistd/execlp.c

«

Si veda la sezione u0.20.

```
3870001 #include <unistd.h>
3870002 #include <string.h>
3870003 #include <stdlib.h>
3870004 #include <errno.h>
3870005 #include <sys/os16.h>
3870006 //-------------------------------------------------------------------------
3870007 int
3870008 execlp (const char *path, const char *arg, ...)
3870009 {
3870010     int   argc;
3870011     char *arg_next;
3870012     char *argv[ARG_MAX/2];
3870013     char  command[PATH_MAX];
3870014     int   status;
3870015     //
3870016     va_list ap;
3870017     va_start (ap, arg);
3870018     //
3870019     arg_next = arg;
3870020     //
3870021     for (argc = 0; argc < ARG_MAX/2; argc++)
3870022       {
3870023         argv[argc] = arg_next;
3870024         if (argv[argc] == NULL)
3870025           {
3870026             break;      // End of arguments.
3870027           }
3870028         arg_next = va_arg (ap, char *);
3870029       }
3870030     //
3870031     // Get a full command path if necessary.
3870032     //
3870033     status = namep (path, command, (size_t) PATH_MAX);
3870034     if (status != 0)
3870035       {
3870036         //
3870037         // Variable ‘errno’ is already set by ‘commandp()’.
3870038         //
3870039         return (-1);
3870040       }
3870041     //
3870042     // Return calling ‘execve()’
3870043     //
3870044     return (execve (command, argv, environ));   // [1]
3870045 }
3870046 //
3870047 // The variable ‘environ’ is declared as ‘char **environ’ and is
```

```
// included from <unistd.h>.
//
```

lib/unistd/execv.c

«

Si veda la sezione u0.20.

```
#include <unistd.h>
//------------------------------------------------------------------------
int
execv (const char *path, char *const argv[])
{
    return (execve (path, argv, environ));      // [1]
}
//
// The variable ‘environ’ is declared as ‘char **environ’ and is
// included from <unistd.h>.
//
```

lib/unistd/execve.c

«

Si veda la sezione u0.10.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os16.h>
#include <errno.h>
#include <string.h>
#include <string.h>
//------------------------------------------------------------------------
int
execve (const char *path, char *const argv[], char *const envp[])
{
    sysmsg_exec_t msg;
    size_t         size;
    size_t         arg_size;
    int            argc;
    size_t         env_size;
    int            envc;
    char          *arg_data = msg.arg_data;
    char          *env_data = msg.env_data;
    //
    msg.ret   = 0;
    msg.errno = 0;
    //
    strncpy (msg.path, path, PATH_MAX);
    //
    // Copy ‘argv[]’ inside a the message buffer ‘msg.arg_data’,
    // separating each string with a null character and counting the
    // number of strings inside ‘argc’.
    //
    for (argc = 0, arg_size = 0, size = 0;
         argv       != NULL         &&
         argc        < (ARG_MAX/16) &&
         arg_size    < ARG_MAX/2    &&
         argv[argc] != NULL;
         argc++, arg_size += size)
      {
        size = strlen (argv[argc]);
        size++;         // Count also the final null character.
        if (size > (ARG_MAX/2 - arg_size))
          {
            errset (E2BIG);     // Argument list too long.
            return (-1);
          }
        strncpy (arg_data, argv[argc], size);
        arg_data += size;
      }
    msg.argc = argc;
    //
    // Copy ‘envp[]’ inside a the message buffer ‘msg.env_data’,
    // separating each string with a null character and counting the
    // number of strings inside ‘envc’.
    //
    for (envc = 0, env_size = 0, size = 0;
         envp       != NULL         &&
         envc        < (ARG_MAX/16) &&
         env_size    < ARG_MAX/2    &&
         envp[envc] != NULL;
         envc++, env_size += size)
      {
        size = strlen (envp[envc]);
        size++;         // Count also the final null character.
        if (size > (ARG_MAX/2 - env_size))
          {
            errset (E2BIG);     // Argument list too long.
            return (-1);
          }
        strncpy (env_data, envp[envc], size);
        env_data += size;
      }
    msg.envc = envc;
    //
    // System call.
    //
    sys (SYS_EXEC, &msg, (sizeof msg));
    //
    // Should not return, but if it does, then there is an error.
    //
    errno = msg.errno;
```



```
    errln = msg.errln;
    strncpy (errfn, msg.errfn, PATH_MAX);
    return (msg.ret);
}
```

lib/unistd/execvp.c

«

Si veda la sezione u0.20.

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/os16.h>
//------------------------------------------------------------------------
int
execvp (const char *path, char *const argv[])
{
    char  command[PATH_MAX];
    int   status;
    //
    // Get a full command path if necessary.
    //
    status = namep (path, command, (size_t) PATH_MAX);
    if (status != 0)
      {
        //
        // Variable ‘errno’ is already set by ‘namep()’.
        //
        return (-1);
      }
    //
    // Return calling ‘execve()’
    //
    return (execve (command, argv, environ));   // [1]
}
//
// The variable ‘environ’ is declared as ‘char **environ’ and is
// included from <unistd.h>.
//
```

lib/unistd/fchdir.c

«

Si veda la sezione u0.2.

```
#include <unistd.h>
#include <errno.h>
//------------------------------------------------------------------------
int
fchdir (int fdn)
{
    //
    // os16 requires to keep track of the path for the current working
    // directory. The standard function ‘fchdir()’ is not applicable.
    //
    errset (E_NOT_IMPLEMENTED);
    return (-1);
}
```

lib/unistd/fchown.c

«

Si veda la sezione u0.5.

```
#include <unistd.h>
#include <string.h>
#include <const.h>
#include <sys/os16.h>
#include <errno.h>
#include <limits.h>
//------------------------------------------------------------------------
int
fchown (int fdn, uid_t uid, gid_t gid)
{
    sysmsg_fchown_t msg;
    //
    msg.fdn = fdn;
    msg.uid = uid;
    msg.gid = gid;
    //
    sys (SYS_FCHOWN, &msg, (sizeof msg));
    //
    errno = msg.errno;
    errln = msg.errln;
    strncpy (errfn, msg.errfn, PATH_MAX);
    return (msg.ret);
}
```

lib/unistd/fork.c

«

Si veda la sezione u0.14.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os16.h>
#include <errno.h>
#include <string.h>
//------------------------------------------------------------------------
```

```
3930007 pid_t
3930008 fork (void)
3930009 {
3930010     sysmsg_fork_t msg;
3930011     //
3930012     // Set the return value for the child process.
3930013     //
3930014     msg.ret = 0;
3930015     //
3930016     // Do the system call.
3930017     //
3930018     sys (SYS_FORK, &msg, (sizeof msg));
3930019     //
3930020     // If the system call has successfully generated a copy of
3930021     // the original process, the following code is executed from
3930022     // the parent and the child. But the child has the 'msg'
3930023     // structure untouched, while the parent has, at least, the
3930024     // pid number inside 'msg.ret'.
3930025     // If the system call fails, there is no child, and the
3930026     // parent finds the return value equal to -1, with an
3930027     // error number.
3930028     //
3930029     errno = msg.errno;
3930030     errln = msg.errln;
3930031     strncpy (errfn, msg.errfn, PATH_MAX);
3930032     return (msg.ret);
3930033 }
```

lib/unistd/getcwd.c

« Si veda la sezione u0.16.

```
3940001 #include <unistd.h>
3940002 #include <sys/types.h>
3940003 #include <sys/os16.h>
3940004 #include <errno.h>
3940005 #include <stddef.h>
3940006 #include <string.h>
3940007 //---------------------------------------------------------------------
3940008 char *
3940009 getcwd (char *buffer, size_t size)
3940010 {
3940011     sysmsg_uarea_t msg;
3940012     //
3940013     // Check arguments: the buffer must be given.
3940014     //
3940015     if (buffer == NULL)
3940016       {
3940017         errset (EINVAL);
3940018         return (NULL);
3940019       }
3940020     //
3940021     // Make shure that the last character, inside the working directory
3940022     // path is a null character.
3940023     //
3940024     msg.path_cwd[PATH_MAX-1] = 0;
3940025     //
3940026     // Just get the user area data.
3940027     //
3940028     sys (SYS_UAREA, &msg, (sizeof msg));
3940029     //
3940030     // Check that the path is still correctly terminated. If it isn't,
3940031     // the path is longer than the implementation limits, and it is
3940032     // really *bad*.
3940033     //
3940034     if (msg.path_cwd[PATH_MAX-1] != 0)
3940035       {
3940036         errset (E_LIMIT);       // Exceeded implementation limits.
3940037         return (NULL);
3940038       }
3940039     //
3940040     // If the path is larger than the buffer size, return an error.
3940041     // Please note that the parameter 'size' must include the
3940042     // terminating null character, so, if the string is equal to
3940043     // the size, it is already beyond the size limit.
3940044     //
3940045     if (strlen (msg.path_cwd) >= size)
3940046       {
3940047         errset (ERANGE);        // Result too large.
3940048         return (NULL);
3940049       }
3940050     //
3940051     // Everything is fine, so, copy the path to the buffer and return.
3940052     //
3940053     strncpy (buffer, msg.path_cwd, size);
3940054     return (buffer);
3940055 }
```

lib/unistd/geteuid.c

« Si veda la sezione u0.18.

```
3950001 #include <unistd.h>
3950002 #include <sys/types.h>
3950003 #include <sys/os16.h>
3950004 #include <errno.h>
3950005 //---------------------------------------------------------------------
3950006 uid_t
3950007 geteuid (void)
3950008 {
```



```
3950009     sysmsg_uarea_t msg;
3950010     sys (SYS_UAREA, &msg, (sizeof msg));
3950011     return (msg.euid);
3950012 }
```

lib/unistd/getopt.c

« Si veda la sezione u0.52.

```
3960001 #include <unistd.h>
3960002 #include <sys/types.h>
3960003 #include <sys/os16.h>
3960004 #include <errno.h>
3960005 //---------------------------------------------------------------------
3960006 char *optarg;
3960007 int   optind    = 1;
3960008 int   opterr    = 1;
3960009 int   optopt    = 0;
3960010 //---------------------------------------------------------------------
3960011 static void getopt_no_argument (int opt);
3960012 //---------------------------------------------------------------------
3960013 int
3960014 getopt (int argc, char *const argv[], const char *optstring)
3960015 {
3960016     static int o = 0;           // Index to scan grouped options.
3960017            int s;               // Index to scan 'optstring'
3960018            int opt;             // Current option letter.
3960019            int flag_argument;   // If there should be an argument.
3960020     //
3960021     // Entering the function, 'flag_argument' is zero. Just to make
3960022     // it clear:
3960023     //
3960024     flag_argument = 0;
3960025     //
3960026     // Scan 'argv[]' elements, starting form the value that 'optind'
3960027     // already have.
3960028     //
3960029     for (; optind < argc; optind++)
3960030       {
3960031         //
3960032         // If an option is expected, some check must be done at
3960033         // the beginning.
3960034         //
3960035         if (!flag_argument)
3960036           {
3960037             //
3960038             // Check if the scan is finished and 'optind' should be kept
3960039             // untouched:
3960040             //   'argv[optind]'    is a null pointer;
3960041             //   'argv[optind][0]' is not the character '-';
3960042             //   'argv[optind]'    points to the string "-";
3960043             //   all 'argv[]' elements are parsed.
3960044             //
3960045             if (argv[optind] == NULL
3960046              || argv[optind][0] != '-'
3960047              || argv[optind][1] == 0
3960048              || optind >= argc)
3960049               {
3960050                 return (-1);
3960051               }
3960052             //
3960053             // Check if the scan is finished and 'optind' is to be
3960054             // incremented:
3960055             //   'argv[optind]'    points to the string "--".
3960056             //
3960057             if (argv[optind][0] == '-'
3960058              && argv[optind][1] == '-'
3960059              && argv[optind][2] == 0)
3960060               {
3960061                 optind++;
3960062                 return (-1);
3960063               }
3960064           }
3960065         //
3960066         // Scan 'argv[optind]' using the static index 'o'.
3960067         //
3960068         for (; o < strlen (argv[optind]); o++)
3960069           {
3960070             //
3960071             // If there should be an option, index 'o' should
3960072             // start from 1, because 'argv[optind][0]' must
3960073             // be equal to '-'.
3960074             //
3960075             if (!flag_argument && (o == 0))
3960076               {
3960077                 //
3960078                 // As there is no options, 'o' cannot start
3960079                 // from zero, so a new loop is done.
3960080                 //
3960081                 continue;
3960082               }
3960083             //
3960084             if (flag_argument)
3960085               {
3960086                 //
3960087                 // There should be an argument, starting from
3960088                 // 'argv[optind][o]'.
3960089                 //
3960090                 if ((o == 0) && (argv[optind][o] == '-'))
3960091                   {
3960092                     //
```

```
                    // 'argv[optind][0]' is equal to '-', but there
                    // should be an argument instead: the argument
                    // is missing.
                    //
                    optarg = NULL;
                    //
                    if (optstring[0] == ':')
                      {
                        //
                        // As the option string starts with ':' the
                        // function must return ':'.
                        //
                        optopt = opt;
                        opt = ':';
                      }
                    else
                      {
                        //
                        // As the option string does not start with ':'
                        // the function must return '?'.
                        //
                        getopt_no_argument (opt);
                        optopt = opt;
                        opt = '?';
                      }
                    //
                    // 'optind' is left untouched.
                    //
                  }
                else
                  {
                    //
                    // The argument is found: 'optind' is to be
                    // incremented and 'o' is reset.
                    //
                    optarg = &argv[optind][o];
                    optind++;
                    o = 0;
                  }
                //
                // Return the option, or ':', or '?'.
                //
                return (opt);
              }
            else
              {
                //
                // It should be an option: 'optstring[]' must be
                // scanned.
                //
                opt = argv[optind][o];
                //
                for (s = 0, optopt = 0; s < strlen (optstring); s++)
                  {
                    //
                    // If 'optsting[0]' is equal to ':', index 's' must
                    // start at 1.
                    //
                    if ((s == 0) && (optstring[0] == ':'))
                      {
                        continue;
                      }
                    //
                    if (opt == optstring[s])
                      {
                        //
                        if (optstring[s+1] == ':')
                          {
                            //
                            // There is an argument.
                            //
                            flag_argument = 1;
                            break;
                          }
                        else
                          {
                            //
                            // There is no argument.
                            //
                            o++;
                            return (opt);
                          }
                      }
                  }
                //
                if (s >= strlen (optstring))
                  {
                    //
                    // The 'optstring' scan is concluded with no
                    // match.
                    //
                    o++;
                    optopt = opt;
                    return ('?');
                  }
                //
                // Otherwise the loop was broken.
                //
              }
          }
        //
```



```
        // Check index 'o'.
        //
        if (o >= strlen (argv[optind]))
          {
            //
            // There are no more options or there is no argument
            // inside current 'argv[optind]' string. Index 'o' is
            // reset before the next loop.
            //
            o = 0;
          }
      }
    //
    // No more elements inside 'argv' or loop broken: there might be a
    // missing argument.
    //
    if (flag_argument)
      {
        //
        // Missing option argument.
        //
        optarg = NULL;
        //
        if (optstring[0] == ':')
          {
            return (':');
          }
        else
          {
            getopt_no_argument (opt);
            return ('?');
          }
      }
    //
    return (-1);
}
//---------------------------------------------------------------------
static void
getopt_no_argument (int opt)
{
    if (opterr)
      {
        fprintf (stderr, "Missing argument for option '-%c'\n", opt);
      }
}
```

lib/unistd/getpgrp.c

«

Si veda la sezione u0.20.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os16.h>
#include <errno.h>
//---------------------------------------------------------------------
pid_t
getpgrp (void)
{
    sysmsg_uarea_t msg;
    sys (SYS_UAREA, &msg, (sizeof msg));
    return (msg.pgrp);
}
```

lib/unistd/getpid.c

«

Si veda la sezione u0.20.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os16.h>
#include <errno.h>
//---------------------------------------------------------------------
pid_t
getpid (void)
{
    sysmsg_uarea_t msg;
    sys (SYS_UAREA, &msg, (sizeof msg));
    return (msg.pid);
}
```

lib/unistd/getppid.c

«

Si veda la sezione u0.20.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/os16.h>
#include <errno.h>
//---------------------------------------------------------------------
pid_t
getppid (void)
{
    sysmsg_uarea_t msg;
    sys (SYS_UAREA, &msg, (sizeof msg));
    return (msg.ppid);
}
```

lib/unistd/getuid.c

«

Si veda la sezione u0.18.

```
4000001 #include <unistd.h>
4000002 #include <sys/types.h>
4000003 #include <sys/os16.h>
4000004 #include <errno.h>
4000005 //---------------------------------------------------------------------
4000006 uid_t
4000007 getuid (void)
4000008 {
4000009     sysmsg_uarea_t msg;
4000010     sys (SYS_UAREA, &msg, (sizeof msg));
4000011     return (msg.uid);
4000012 }
```

lib/unistd/isatty.c

«

Si veda la sezione u0.61.

```
4010001 #include <sys/stat.h>
4010002 #include <sys/os16.h>
4010003 #include <unistd.h>
4010004 #include <sys/types.h>
4010005 #include <errno.h>
4010006 //---------------------------------------------------------------------
4010007 int
4010008 isatty (int fdn)
4010009 {
4010010     struct stat file_status;
4010011     //
4010012     // Verify to have valid input data.
4010013     //
4010014     if (fdn < 0)
4010015       {
4010016         errset (EBADF);
4010017         return (0);
4010018       }
4010019     //
4010020     // Verify the standard input.
4010021     //
4010022     if (fstat(fdn, &file_status) == 0)
4010023       {
4010024         if (major (file_status.st_rdev) == DEV_CONSOLE_MAJOR)
4010025           {
4010026             return (1); // Meaning it is ok!
4010027           }
4010028         if (major (file_status.st_rdev) == DEV_TTY_MAJOR)
4010029           {
4010030             return (1); // Meaning it is ok!
4010031           }
4010032       }
4010033     else
4010034       {
4010035         errset (errno);
4010036         return (0);
4010037       }
4010038     //
4010039     // If here, it is not a terminal of any kind.
4010040     //
4010041     errset (EINVAL);
4010042     return (0);
4010043 }
```

lib/unistd/link.c

«

Si veda la sezione u0.23.

```
4020001 #include <unistd.h>
4020002 #include <string.h>
4020003 #include <const.h>
4020004 #include <sys/os16.h>
4020005 #include <errno.h>
4020006 #include <limits.h>
4020007 //---------------------------------------------------------------------
4020008 int
4020009 link (const char *path_old, const char *path_new)
4020010 {
4020011     sysmsg_link_t msg;
4020012     //
4020013     strncpy (msg.path_old, path_old, PATH_MAX);
4020014     strncpy (msg.path_new, path_new, PATH_MAX);
4020015     //
4020016     sys (SYS_LINK, &msg, (sizeof msg));
4020017     //
4020018     errno = msg.errno;
4020019     errln = msg.errln;
4020020     strncpy (errfn, msg.errfn, PATH_MAX);
4020021     return (msg.ret);
4020022 }
```

lib/unistd/lseek.c

«

Si veda la sezione u0.24.

```
4030001 #include <unistd.h>
4030002 #include <sys/types.h>
4030003 #include <sys/os16.h>
```



```
4030004 #include <errno.h>
4030005 #include <string.h>
4030006 //---------------------------------------------------------------------
4030007 off_t
4030008 lseek (int fdn, off_t offset, int whence)
4030009 {
4030010     sysmsg_lseek_t  msg;
4030011     msg.fdn    = fdn;
4030012     msg.offset = offset;
4030013     msg.whence = whence;
4030014     sys (SYS_LSEEK, &msg, (sizeof msg));
4030015     errno = msg.errno;
4030016     errln = msg.errln;
4030017     strncpy (errfn, msg.errfn, PATH_MAX);
4030018     return (msg.ret);
4030019 }
```

lib/unistd/read.c

«

Si veda la sezione u0.29.

```
4040001 #include <unistd.h>
4040002 #include <sys/os16.h>
4040003 #include <errno.h>
4040004 #include <string.h>
4040005 #include <stdio.h>
4040006 //---------------------------------------------------------------------
4040007 ssize_t
4040008 read (int fdn, void *buffer, size_t count)
4040009 {
4040010     sysmsg_read_t msg;
4040011     //
4040012     // Reduce size of read if necessary.
4040013     //
4040014     if (count > BUFSIZ)
4040015       {
4040016         count = BUFSIZ;
4040017       }
4040018     //
4040019     // Fill the message.
4040020     //
4040021     msg.fdn   = fdn;
4040022     msg.count = count;
4040023     msg.eof   = 0;
4040024     msg.ret   = 0;
4040025     //
4040026     // Repeat syscall, until something is received or end of file is
4040027     // reached.
4040028     //
4040029     while (1)
4040030       {
4040031         sys (SYS_READ, &msg, (sizeof msg));
4040032         if (msg.ret != 0 || msg.eof)
4040033           {
4040034             break;
4040035           }
4040036       }
4040037     //
4040038     // Before return: be careful with the 'msg.buffer' copy, because
4040039     // it cannot be longer than 'count', otherwise, some unexpected
4040040     // memory will be overwritten!
4040041     //
4040042     if (msg.ret < 0)
4040043       {
4040044         //
4040045         // No valid read, no change inside the buffer.
4040046         //
4040047         errno = msg.errno;
4040048         return (msg.ret);
4040049       }
4040050     //
4040051     if (msg.ret > count)
4040052       {
4040053         //
4040054         // A strange value was returned. Considering it a read error.
4040055         //
4040056         errset (EIO);                   // I/O error.
4040057         return (-1);
4040058       }
4040059     //
4040060     // A valid read: fill the buffer with 'msg.ret' bytes.
4040061     //
4040062     memcpy (buffer, msg.buffer, msg.ret);
4040063     //
4040064     // Return.
4040065     //
4040066     return (msg.ret);
4040067 }
```

lib/unistd/rmdir.c

«

Si veda la sezione u0.30.

```
4050001 #include <unistd.h>
4050002 #include <string.h>
4050003 #include <const.h>
4050004 #include <sys/os16.h>
4050005 #include <errno.h>
4050006 #include <limits.h>
4050007 //---------------------------------------------------------------------
```

```
4050008 int
4050009 rmdir (const char *path)
4050010 {
4050011     sysmsg_stat_t   msg_stat;
4050012     sysmsg_unlink_t msg_unlink;
4050013     //
4050014     if (path == NULL)
4050015       {
4050016         errset (EINVAL);
4050017         return (-1);
4050018       }
4050019     //
4050020     strncpy (msg_stat.path, path, PATH_MAX);
4050021     //
4050022     sys (SYS_STAT, &msg_stat, (sizeof msg_stat));
4050023     //
4050024     if (msg_stat.ret != 0)
4050025       {
4050026         errno = msg_stat.errno;
4050027         errln = msg_stat.errln;
4050028         strncpy (errfn, msg_stat.errfn, PATH_MAX);
4050029         return (msg_stat.ret);
4050030       }
4050031     //
4050032     if (!S_ISDIR (msg_stat.stat.st_mode))
4050033       {
4050034         errset (ENOTDIR);               // Not a directory.
4050035         return (-1);
4050036       }
4050037     //
4050038     strncpy (msg_unlink.path, path, PATH_MAX);
4050039     //
4050040     sys (SYS_UNLINK, &msg_unlink, (sizeof msg_unlink));
4050041     //
4050042     errno = msg_unlink.errno;
4050043     errln = msg_unlink.errln;
4050044     strncpy (errfn, msg_unlink.errfn, PATH_MAX);
4050045     return (msg_unlink.ret);
4050046 }
```

lib/unistd/seteuid.c

«

Si veda la sezione u0.33.

```
4060001 #include <unistd.h>
4060002 #include <sys/types.h>
4060003 #include <sys/os16.h>
4060004 #include <errno.h>
4060005 #include <string.h>
4060006 //-------------------------------------------------------------------------
4060007 int
4060008 seteuid (uid_t uid)
4060009 {
4060010     sysmsg_seteuid_t msg;
4060011     msg.ret   = 0;
4060012     msg.errno = 0;
4060013     msg.euid  = uid;
4060014     sys (SYS_SETEUID, &msg, (sizeof msg));
4060015     errno = msg.errno;
4060016     errln = msg.errln;
4060017     strncpy (errfn, msg.errfn, PATH_MAX);
4060018     return (msg.ret);
4060019 }
4060020
```

lib/unistd/setpgrp.c

«

Si veda la sezione u0.32.

```
4070001 #include <unistd.h>
4070002 #include <sys/os16.h>
4070003 #include <stddef.h>
4070004 //-------------------------------------------------------------------------
4070005 int
4070006 setpgrp (void)
4070007 {
4070008     sys (SYS_PGRP, NULL, (size_t) 0);
4070009     return (0);
4070010 }
```

lib/unistd/setuid.c

«

Si veda la sezione u0.33.

```
4080001 #include <unistd.h>
4080002 #include <sys/types.h>
4080003 #include <sys/os16.h>
4080004 #include <errno.h>
4080005 #include <string.h>
4080006 //-------------------------------------------------------------------------
4080007 int
4080008 setuid (uid_t uid)
4080009 {
4080010     sysmsg_setuid_t msg;
4080011     msg.ret   = 0;
4080012     msg.errno = 0;
4080013     msg.euid  = uid;
4080014     sys (SYS_SETUID, &msg, (sizeof msg));
```



```
4080015     errno = msg.errno;
4080016     errln = msg.errln;
4080017     strncpy (errfn, msg.errfn, PATH_MAX);
4080018     return (msg.ret);
4080019 }
```

lib/unistd/sleep.c

«

Si veda la sezione u0.35.

```
4090001 #include <unistd.h>
4090002 #include <sys/types.h>
4090003 #include <sys/os16.h>
4090004 #include <errno.h>
4090005 #include <time.h>
4090006 //-------------------------------------------------------------------------
4090007 unsigned int
4090008 sleep (unsigned int seconds)
4090009 {
4090010     sysmsg_sleep_t  msg;
4090011     time_t          start;
4090012     time_t          end;
4090013     int             slept;
4090014     //
4090015     if (seconds == 0)
4090016       {
4090017         return (0);
4090018       }
4090019     //
4090020     msg.events  = WAKEUP_EVENT_TIMER;
4090021     msg.seconds = seconds;
4090022     sys (SYS_SLEEP, &msg, (sizeof msg));
4090023     start = msg.ret;
4090024     end   = time (NULL);
4090025     slept = end - msg.ret;
4090026     //
4090027     if (slept < 0)
4090028       {
4090029         return (seconds);
4090030       }
4090031     else if (slept < seconds)
4090032       {
4090033         return (seconds - slept);
4090034       }
4090035     else
4090036       {
4090037         return (0);
4090038       }
4090039 }
```

lib/unistd/ttyname.c

«

Si veda la sezione u0.124.

```
4100001 #include <sys/os16.h>
4100002 #include <sys/stat.h>
4100003 #include <unistd.h>
4100004 #include <sys/types.h>
4100005 #include <errno.h>
4100006 #include <limits.h>
4100007 //-------------------------------------------------------------------------
4100008 char *
4100009 ttyname (int fdn)
4100010 {
4100011     int         dev_minor;
4100012     struct stat file_status;
4100013     static char name[PATH_MAX];
4100014     //
4100015     // Verify to have valid input data.
4100016     //
4100017     if (fdn < 0)
4100018       {
4100019         errset (EBADF);
4100020         return (NULL);
4100021       }
4100022     //
4100023     // Verify the file descriptor.
4100024     //
4100025     if (fstat (fdn, &file_status) == 0)
4100026       {
4100027         if (major (file_status.st_rdev) == DEV_CONSOLE_MAJOR)
4100028           {
4100029             dev_minor = minor (file_status.st_rdev);
4100030             //
4100031             // If minor is equal to 0xFF, it is '/dev/console'.
4100032             //
4100033             if (dev_minor < 0xFF)
4100034               {
4100035                 sprintf (name, "/dev/console%i", dev_minor);
4100036               }
4100037             else
4100038               {
4100039                 strcpy (name, "/dev/console");
4100040               }
4100041             return (name);
4100042           }
4100043         else if (file_status.st_rdev == DEV_TTY)
4100044           {
4100045             strcpy (name, "/dev/tty");
4100046             return (name);
```

```
4100047             }
4100048           else
4100049             {
4100050               errset (ENOTTY);
4100051               return (NULL);
4100052             }
4100053         }
4100054       else
4100055         {
4100056           errset (errno);
4100057           return (NULL);
4100058         }
4100059 }
```

lib/unistd/unlink.c

«

Si veda la sezione u0.42.

```
4110001 #include <unistd.h>
4110002 #include <string.h>
4110003 #include <const.h>
4110004 #include <sys/os16.h>
4110005 #include <errno.h>
4110006 #include <limits.h>
4110007 //------------------------------------------------------------------------
4110008 int
4110009 unlink (const char *path)
4110010 {
4110011     sysmsg_unlink_t msg;
4110012     //
4110013     strncpy (msg.path, path, PATH_MAX);
4110014     //
4110015     sys (SYS_UNLINK, &msg, (sizeof msg));
4110016     //
4110017     errno = msg.errno;
4110018     errln = msg.errln;
4110019     strncpy (errfn, msg.errfn, PATH_MAX);
4110020     return (msg.ret);
4110021 }
```

lib/unistd/write.c

«

Si veda la sezione u0.44.

```
4120001 #include <unistd.h>
4120002 #include <sys/os16.h>
4120003 #include <errno.h>
4120004 #include <string.h>
4120005 #include <const.h>
4120006 #include <stdio.h>
4120007 //------------------------------------------------------------------------
4120008 ssize_t
4120009 write (int fdn, const void *buffer, size_t count)
4120010 {
4120011     sysmsg_write_t msg;
4120012     //
4120013     // Reduce size of write if necessary.
4120014     //
4120015     if (count > BUFSIZ)
4120016       {
4120017         count = BUFSIZ;
4120018       }
4120019     //
4120020     // Fill the message.
4120021     //
4120022     msg.fdn   = fdn;
4120023     msg.count = count;
4120024     memcpy (msg.buffer, buffer, count);
4120025     //
4120026     // Syscall.
4120027     //
4120028     sys (SYS_WRITE, &msg, (sizeof msg));
4120029     //
4120030     // Check result and return.
4120031     //
4120032     if (msg.ret < 0)
4120033       {
4120034         //
4120035         // No valid read, no change inside the buffer.
4120036         //
4120037         errno = msg.errno;
4120038         errln = msg.errln;
4120039         strncpy (errfn, msg.errfn, PATH_MAX);
4120040         return (msg.ret);
4120041       }
4120042     //
4120043     if (msg.ret > count)
4120044       {
4120045         //
4120046         // A strange value was returned. Considering it a read error.
4120047         //
4120048         errset (EIO);                   // I/O error.
4120049         return (-1);
4120050       }
4120051     //
4120052     // A valid write return.
4120053     //
4120054     return (msg.ret);
4120055 }
```


os16: «lib/utime.h»

«

Si veda la sezione u0.2.

```
4130001 #ifndef _UTIME_H
4130002 #define _UTIME_H        1
4130003
4130004 #include <const.h>
4130005 #include <restrict.h>
4130006 #include <sys/types.h>          // time_t
4130007
4130008 //------------------------------------------------------------------------
4130009 struct utimbuf {
4130010     time_t  actime;
4130011     time_t  modtime;
4130012 };
4130013 //------------------------------------------------------------------------
4130014 int utime (const char *path, const struct utimbuf *times);
4130015 //------------------------------------------------------------------------
4130016
4130017 #endif
```

lib/utime/utime.c

«

Si veda la sezione u0.2.

```
4140001 #include <utime.h>
4140002 #include <errno.h>
4140003 //------------------------------------------------------------------------
4140004 int
4140005 utime (const char *path, const struct utimbuf *times)
4140006 {
4140007     //
4140008     // Currently not implemented.
4140009     //
4140010     return (0);
4140011 }
```

os16: directory «applic/» 1850
applic/MAKEDEV.c 1850
applic/aaa.c 1850
applic/bbb.c 1851
applic/cat.c 1851
applic/ccc.c 1852
applic/chmod.c 1852
applic/chown.c 1853
applic/cp.c 1854
applic/crt0.s 1856
applic/date.c 1857
applic/ed.c 1859
applic/getty.c 1873
applic/init.c 1874
applic/kill.c 1876
applic/ln.c 1879
applic/login.c 1880
applic/ls.c 1882
applic/man.c 1885
applic/mkdir.c 1888
applic/more.c 1890
applic/mount.c 1892
applic/ps.c 1893
applic/rm.c 1894
applic/shell.c 1894
applic/touch.c 1897
applic/tty.c 1898
applic/umount.c 1898

aaa.c 1850 bbb.c 1851 cat.c 1851 ccc.c 1852 chmod.c 1852 chown.c 1853 cp.c 1854 crt0.s 1856 date.c 1857 ed.c 1859 getty.c 1873 init.c 1874 kill.c 1876 ln.c 1879 login.c 1880 ls.c 1882 MAKEDEV.c 1850 man.c 1885 mkdir.c 1888 more.c 1890 mount.c 1892 ps.c 1893 rm.c 1894 shell.c 1894 touch.c 1897 tty.c 1898 umount.c 1898

os16: directory «applic/» 1850
applic/MAKEDEV.c 1850
applic/aaa.c 1850
applic/bbb.c 1851
applic/cat.c 1851
applic/ccc.c 1852
applic/chmod.c 1852
applic/chown.c 1853
applic/cp.c 1854
applic/crt0.s 1856
applic/date.c 1857
applic/ed.c 1859
applic/getty.c 1873
applic/init.c 1874
applic/kill.c 1876
applic/ln.c 1879
applic/login.c 1880
applic/ls.c 1882
applic/man.c 1885
applic/mkdir.c 1888
applic/more.c 1890

applic/mount.c 1892
applic/ps.c 1893
applic/rm.c 1894
applic/shell.c 1894
applic/touch.c 1897
applic/tty.c 1898
applic/umount.c 1898

os16: directory «applic/»

«

applic/MAKEDEV.c

«

Si veda la sezione u0.3.

```
4150001 #include <unistd.h>
4150002 #include <stdlib.h>
4150003 #include <sys/stat.h>
4150004 #include <fcntl.h>
4150005 #include <kernel/devices.h>
4150006 #include <stdio.h>
4150007 //------------------------------------------------------------------------
4150008 int
4150009 main (void)
4150010 {
4150011     int status;
4150012     status = mknod ("mem",        (mode_t) (S_IFCHR | 0444),
4150013                     (dev_t) DEV_MEM);
4150014     if (status) perror (NULL);
4150015     status = mknod ("null",       (mode_t) (S_IFCHR | 0666),
4150016                     (dev_t) DEV_NULL);
4150017     if (status) perror (NULL);
4150018     status = mknod ("port",       (mode_t) (S_IFCHR | 0644),
4150019                     (dev_t) DEV_PORT);
4150020     if (status) perror (NULL);
4150021     status = mknod ("zero",       (mode_t) (S_IFCHR | 0666),
4150022                     (dev_t) DEV_ZERO);
4150023     if (status) perror (NULL);
4150024     status = mknod ("tty",        (mode_t) (S_IFCHR | 0666),
4150025                     (dev_t) DEV_TTY);
4150026     if (status) perror (NULL);
4150027     status = mknod ("dsk0",       (mode_t) (S_IFBLK | 0644),
4150028                     (dev_t) DEV_DSK0);
4150029     if (status) perror (NULL);
4150030     status = mknod ("dsk1",       (mode_t) (S_IFBLK | 0644),
4150031                     (dev_t) DEV_DSK1);
4150032     if (status) perror (NULL);
4150033     status = mknod ("dsk2",       (mode_t) (S_IFBLK | 0644),
4150034                     (dev_t) DEV_DSK2);
4150035     if (status) perror (NULL);
4150036     status = mknod ("dsk3",       (mode_t) (S_IFBLK | 0644),
4150037                     (dev_t) DEV_DSK3);
4150038     if (status) perror (NULL);
4150039     status = mknod ("kmem_ps",    (mode_t) (S_IFCHR | 0444),
4150040                     (dev_t) DEV_KMEM_PS);
4150041     if (status) perror (NULL);
4150042     status = mknod ("kmem_mmp",   (mode_t) (S_IFCHR | 0444),
4150043                     (dev_t) DEV_KMEM_MMP);
4150044     if (status) perror (NULL);
4150045     status = mknod ("kmem_sb",    (mode_t) (S_IFCHR | 0444),
4150046                     (dev_t) DEV_KMEM_SB);
4150047     if (status) perror (NULL);
4150048     status = mknod ("kmem_inode", (mode_t) (S_IFCHR | 0444),
4150049                     (dev_t) DEV_KMEM_INODE);
4150050     if (status) perror (NULL);
4150051     status = mknod ("kmem_file",  (mode_t) (S_IFCHR | 0444),
4150052                     (dev_t) DEV_KMEM_FILE);
4150053     if (status) perror (NULL);
4150054     status = mknod ("console",    (mode_t) (S_IFCHR | 0644),
4150055                     (dev_t) DEV_CONSOLE);
4150056     if (status) perror (NULL);
4150057     status = mknod ("console0",   (mode_t) (S_IFCHR | 0644),
4150058                     (dev_t) DEV_CONSOLE0);
4150059     if (status) perror (NULL);
4150060     status = mknod ("console1",   (mode_t) (S_IFCHR | 0644),
4150061                     (dev_t) DEV_CONSOLE1);
4150062     if (status) perror (NULL);
4150063     status = mknod ("console2",   (mode_t) (S_IFCHR | 0644),
4150064                     (dev_t) DEV_CONSOLE2);
4150065     if (status) perror (NULL);
4150066     status = mknod ("console3",   (mode_t) (S_IFCHR | 0644),
4150067                     (dev_t) DEV_CONSOLE3);
4150068     if (status) perror (NULL);
4150069
4150070     return (0);
4150071 }
```

applic/aaa.c

«

Si veda la sezione u0.1.

```
4160001 #include <unistd.h>
4160002 #include <stdio.h>
4160003 //------------------------------------------------------------------------
4160004 int
4160005 main (void)
4160006 {
4160007     unsigned int count;
4160008     for (count = 0; count < 60; count++)
4160009       {
4160010         printf ("a");
4160011         sleep (1);
4160012       }
4160013     return (8);
4160014 }
```


applic/bbb.c

«

Si veda la sezione u0.1.

```
4170001 #include <unistd.h>
4170002 #include <stdio.h>
4170003 #include <stdlib.h>
4170004 //------------------------------------------------------------------------
4170005 int
4170006 main (void)
4170007 {
4170008     unsigned int count;
4170009     for (count = 0; count < 30; count++)
4170010       {
4170011         printf ("b");
4170012         sleep (2);
4170013       }
4170014     exit (0);
4170015     return (0);
4170016 }
```

applic/cat.c

«

Si veda la sezione u0.3.

```
4180001 #include <fcntl.h>
4180002 #include <sys/stat.h>
4180003 #include <stddef.h>
4180004 #include <unistd.h>
4180005 #include <stdio.h>
4180006 #include <stdlib.h>
4180007 #include <errno.h>
4180008 //------------------------------------------------------------------------
4180009 static void cat_file_descriptor (int fd);
4180010 //------------------------------------------------------------------------
4180011 int
4180012 main (int argc, char *argv[], char *envp[])
4180013 {
4180014     int         i;
4180015     int         fd;
4180016     struct stat file_status;
4180017     //
4180018     // Check if the input comes from standard input.
4180019     //
4180020     if (argc < 2)
4180021       {
4180022         cat_file_descriptor (STDIN_FILENO);
4180023         exit (0);
4180024       }
4180025     //
4180026     // There is at least an argument: scan them.
4180027     //
4180028     for(i = 1; i < argc; i++)
4180029       {
4180030         //
4180031         // Verify if the file exists.
4180032         //
4180033         if (stat(argv[i], &file_status) != 0)
4180034           {
4180035             fprintf (stderr, "File \"%s\" does not exist!\n",
4180036                      argv[i]);
4180037             continue;
4180038           }
4180039         //
4180040         // File exists: check the file type.
4180041         //
4180042         if (S_ISDIR (file_status.st_mode))
4180043           {
4180044             fprintf (stderr, "Cannot \"cat\" "
4180045                              "\"%s\": it is a directory!\n",
4180046                      argv[i]);
4180047             continue;
4180048           }
4180049         //
4180050         // File exists and can be "cat"ed.
4180051         //
4180052         fd = open (argv[i], O_RDONLY);
4180053         if (fd >= 0)
4180054           {
4180055             cat_file_descriptor (fd);
4180056             close (fd);
4180057           }
4180058         else
4180059           {
4180060             perror (NULL);
4180061             exit (1);
4180062           }
4180063       }
4180064     return (0);
```

```
}
//-----------------------------------------------------------------------
static void
cat_file_descriptor (int fd)
{
    ssize_t count;
    char buffer[BUFSIZ];

    for (;;)
      {
        count = read (fd, buffer, (size_t) BUFSIZ);
        if (count > 0)
          {
            write (STDOUT_FILENO, buffer, (size_t) count);
          }
        else
          {
            break;
          }
      }
}

```

applic/ccc.c

«

Si veda la sezione u0.1.

```
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
//-----------------------------------------------------------------------
int
main (void)
{
    pid_t        pid;
    //-----------------------------------------------------------------------
    pid = fork ();
    if (pid == 0)
      {
        setuid ((uid_t) 10);
        execve ("/bin/aaa", NULL, NULL);
        exit (0);
      }
    //-----------------------------------------------------------------------
    pid = fork ();
    if (pid == 0)
      {
        setuid ((uid_t) 11);
        execve ("/bin/bbb", NULL, NULL);
        exit (0);
      }
    //-----------------------------------------------------------------------
    while (1)
      {
        ; // Just loop, to consume CPU time: it must be killed manually.
      }
    return (0);
}
```

applic/chmod.c

«

Si veda la sezione u0.5.

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <sys/os16.h>
//-----------------------------------------------------------------------
static void usage (void);
//-----------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    int     status;
    mode_t  mode;
    char   *m;          // Pointer inside the octal mode string.
    int     digit;
    int     a;          // Argument index.
    //
    //
    //
    if (argc < 3)
      {
        usage ();
        return (1);
      }
    //
    // Get mode: must be the first argument.
    //
    for (m = argv[1]; *m != 0; m++)
      {
        digit = (*m - '0');
        if (digit < 0 || digit > 7)
          {
            usage ();
            return (2);
          }
        mode = mode * 8 + digit;
      }
    //
    // System call for all the remaining arguments.
    //
    for (a = 2; a < argc; a++)
      {
        status = chmod (argv[a], mode);
        if (status != 0)
          {
            perror (argv[a]);
            return (3);
          }
      }
    //
    // All done.
    //
    return (0);
}
//-----------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage:   chmod OCTAL_MODE FILE...\n");
    fprintf (stderr, "Example: chmod 0640 my_file\n");
}
```

applic/chown.c

«

Si veda la sezione u0.6.

```
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <ctype.h>
#include <pwd.h>
//-----------------------------------------------------------------------
static void usage (void);
//-----------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    char            *user;
    int              uid;
    struct passwd   *pws;
    struct stat      file_status;
    int              a;          // Argument index.
    int              status;
    //
    //
    //
    if (argc < 3)
      {
        usage ();
        return (1);
      }
    //
    // Get user id number.
    //
    user = argv[1];
    if (isdigit (*user))
      {
        uid = atoi (user);
      }
    else
      {
        pws = getpwnam (user);
        if (pws == NULL)
          {
            fprintf(stderr, "Unknown user \"%s\"!\n", user);
            return(2);
          }
        uid = pws->pw_uid;
      }
    //
    // Now we have the user id. Start scanning file names.
    //
    for (a = 2; a < argc; a++)
      {
        //
        // Verify if the file exists, through the return value of
        // 'stat()'. No other checks are made.
        //
        if (stat(argv[a], &file_status) == 0)
         {
            //
            // Try to change ownership.
            //
            status = chown (argv[a], uid, file_status.st_gid);
            if (status != 0)
              {
```

```
                perror (NULL);
                return (3);
              }
          }
        else
          {
            fprintf (stderr, "File \"%s\" does not exist!\n",
                     argv[a]);
            continue;
          }
      }
    //
    // All done.
    //
    return (0);
}
//-----------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage:   chown USER|UID FILE...\n");
    fprintf (stderr, "Example: chown user my_file\n");
}
```

applic/cp.c

«

Si veda la sezione u0.7.

```
#include <sys/os16.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
//-----------------------------------------------------------------------
static void usage (void);
//-----------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    char       *source;
    char       *destination;
    char       *destination_full;
    struct stat file_status;
    int         dest_is_a_dir = 0;
    int         a;                      // Argument index.
    char        path[PATH_MAX];
    int         fd_source      = -1;
    int         fd_destination = -1;
    char        buffer_in[BUFSIZ];
    char       *buffer_out;
    ssize_t     count_in;             // Read counter.
    ssize_t     count_out;            // Write counter.
    //
    // There must be at least two arguments, plus the program name.
    //
    if (argc < 3)
      {
        usage ();
        return (1);
      }
    //
    // Select the last argument as the destination.
    //
    destination = argv[argc-1];
    //
    // Check if it is a directory and save it in a flag.
    //
    if (stat (destination, &file_status) == 0)
      {
        if (S_ISDIR (file_status.st_mode))
          {
            dest_is_a_dir = 1;
          }
      }
    //
    // If there are more than two arguments, verify that the last
    // one is a directory.
    //
    if (argc > 3)
      {
        if (!dest_is_a_dir)
          {
            usage ();
            fprintf (stderr, "The destination \"%s\" ",
                     destination);
            fprintf (stderr, "is not a directory!\n");
            return (1);
          }
      }
    //
    // Scan the arguments, excluded the last, that is the destination.
    //
    for (a = 1; a < (argc - 1); a++)
      {
```



```
        //
        // Source.
        //
        source = argv[a];
        //
        // Verify access permissions.
        //
        if (access (source, R_OK) < 0)
          {
            perror (source);
            continue;
          }
        //
        // Destination.
        //
        // If it is a directory, the destination path
        // must be corrected.
        //
        if (dest_is_a_dir)
          {
            path[0] = 0;
            strcat (path, destination);
            strcat (path, "/");
            strcat (path, basename (source));
            //
            // Update the destination path.
            //
            destination_full = path;
          }
        else
          {
            destination_full = destination;
          }
        //
        // Check if destination file exists.
        //
        if (stat (destination_full, &file_status) == 0)
          {
            fprintf (stderr, "The destination file, \"%s\", ",
                     destination_full);
            fprintf (stderr, "already exists!\n");
            continue;
          }
        //
        // Everything is ready for the copy.
        //
        fd_source = open (source, O_RDONLY);
        if (fd_source < 0)
          {
            perror (source);
            //
            // Continue with the next file.
            //
            continue;
          }
        //
        fd_destination = creat (destination_full, 0777);
        if (fd_destination < 0)
          {
            perror (destination);
            close (fd_source);
            //
            // Continue with the next file.
            //
            continue;
          }
        //
        // Copy the data.
        //
        while (1)
          {
            count_in = read (fd_source, buffer_in, (size_t) BUFSIZ);
            if (count_in > 0)
              {
                for (buffer_out = buffer_in; count_in > 0;)
                  {
                    count_out = write (fd_destination, buffer_out,
                                       (size_t) count_in);
                    if (count_out < 0)
                      {
                        perror (destination);
                        close (fd_source);
                        close (fd_destination);
                        return (3);
                      }
                    //
                    // If not all data is written, continue writing,
                    // but change the buffer start position and the
                    // amount to be written.
                    //
                    buffer_out += count_out;
                    count_in -= count_out;
                  }
              }
            else if (count_in < 0)
              {
                perror (source);
                close (fd_source);
                close (fd_destination);
              }
            else
```

```
4220175                 {
4220176                   break;
4220177                 }
4220178             }
4220179           //
4220180           if (close (fd_source))
4220181             {
4220182               perror (source);
4220183             }
4220184           if (close (fd_destination))
4220185             {
4220186               perror (destination);
4220187               return (4);
4220188             }
4220189         }
4220190       //
4220191       // All done.
4220192       //
4220193       return (0);
4220194   }
4220195   //---------------------------------------------------------------------
4220196   static void
4220197   usage (void)
4220198   {
4220199       fprintf (stderr, "Usage: cp OLD_NAME NEW_NAME\n");
4220200       fprintf (stderr, "       cp FILE... DIRECTORY\n");
4220201   }
```

applic/crt0.s

«

Si veda la sezione u0.2.

```
4230001   .extern _main
4230002   .extern __stdio_stream_setup
4230003   .extern __dirent_directory_stream_setup
4230004   .extern __atexit_setup
4230005   .extern __environment_setup
4230006   .global ___mkargv
4230007   ;---------------------------------------------------------------------
4230008   ; Please note that, all segments are already set from the scheduler,
4230009   ; and there is also data inside the stack, so that the call to 'main()'
4230010   ; function will result as expected.
4230011   ;
4230012   ; This is a modified version of 'crt0.s' with a smaller stack size.
4230013   ;---------------------------------------------------------------------
4230014   ; The following statement says that the code will start at "startup"
4230015   ; label.
4230016   ;---------------------------------------------------------------------
4230017   entry startup
4230018   ;---------------------------------------------------------------------
4230019   .text
4230020   ;---------------------------------------------------------------------
4230021   startup:
4230022       ;
4230023       ; Jump after initial data.
4230024       ;
4230025       jmp startup_code
4230026       ;
4230027   filler:
4230028       ;
4230029       ; After four bytes, from the start, there is the
4230030       ; magic number and other data.
4230031       ;
4230032       .space (0x0004 - (filler - startup))
4230033       ;
4230034   magic:
4230035       .data4 0x6F733136   ; os16
4230036       .data4 0x6170706C   ; appl
4230037       ;
4230038   segoff:                 ;
4230039       .data2 __segoff     ; Data segment offset.
4230040   etext:                  ;
4230041       .data2 __etext      ; End of code
4230042   edata:                  ;
4230043       .data2 __edata      ; End of initialized data.
4230044   ebss:                   ;
4230045       .data2 __end        ; End of not initialized data.
4230046   stack_size:             ;
4230047       .data2 0x2000       ; Requested stack size. Every single application
4230048                           ; might change this value.
4230049       ;
4230050       ; At the next label, the work begins.
4230051       ;
4230052   .align 2
4230053   startup_code:
4230054       ;
4230055       ; Before the call to the main function, it is necessary to extract
4230056       ; the value to assign to the global variable 'environ'. It is
4230057       ; described as 'char **environ' and should contain the same address
4230058       ; pointed by 'envp'. To get this value, the stack is popped and then
4230059       ; pushed again. Please recall that the stack was prepared from
4230060       ; the process management, at the 'exec()' system call.
4230061       ;
4230062       pop ax              ; argc
4230063       pop bx              ; argv
4230064       pop cx              ; envp
4230065       mov _environ, cx    ; Variable 'environ' comes from <unistd.h>.
4230066       push cx
4230067       push bx
4230068       push ax
4230069       ;
```



```
4230070       ; Could it be enough? Of course not! To be able to handle the
4230071       ; environment, it must be copied inside the table
4230072       ; '_environment_table[][]', that is defined inside <stdlib.h>.
4230073       ; To copy the environment it is used the function
4230074       ; '_environment_setup()', passing the 'envp' pointer.
4230075       ;
4230076       push cx
4230077       call __environment_setup
4230078       add  sp, #2
4230079       ;
4230080       ; After the environment copy is done, the value for the traditional
4230081       ; variable 'environ' is updated, to point to the new array of
4230082       ; pointer. The updated value comes from variable '_environment',
4230083       ; defined inside <stdlib.h>. Then, also the 'argv' contained inside
4230084       ; the stack is replaced with the new value.
4230085       ;
4230086       mov  ax, #__environment
4230087       mov  _environ, ax
4230088       ;
4230089       pop ax              ; argc
4230090       pop bx              ; argv[][]
4230091       pop cx              ; envp[][]
4230092       mov cx, #__environment
4230093       push cx
4230094       push bx
4230095       push ax
4230096       ;
4230097       ; Setup standard I/O streams and at-exit table.
4230098       ;
4230099       call __stdio_stream_setup
4230100       call __dirent_directory_stream_setup
4230101       call __atexit_setup
4230102       ;
4230103       ; Call the main function. The arguments are already pushed inside
4230104       ; the stack.
4230105       ;
4230106       call _main
4230107       ;
4230108       ; Save the return value at the symbol 'exit_value'.
4230109       ;
4230110       mov  exit_value, ax
4230111       ;
4230112   .align 2
4230113   halt:
4230114       ;
4230115       push #2             ; Size of message.
4230116       push #exit_value    ; Pointer to the message.
4230117       push #6             ; SYS_EXIT
4230118       call _sys
4230119       add  sp, #2
4230120       add  sp, #2
4230121       add  sp, #2
4230122       ;
4230123       jmp halt
4230124       ;
4230125   ;---------------------------------------------------------------------
4230126   .align 2
4230127   ___mkargv:      ; Symbol '___mkargv' is used by Bcc inside the function
4230128       ret         ; 'main()' and must be present for a successful
4230129                   ; compilation.
4230130   ;---------------------------------------------------------------------
4230131   .align 2
4230132   .data
4230133   ;
4230134   exit_value:
4230135       .data2 0x0000
4230136   ;---------------------------------------------------------------------
4230137   .align 2
4230138   .bss
```

applic/date.c

«

Si veda la sezione u0.8.

```
4240001   #include <unistd.h>
4240002   #include <stdlib.h>
4240003   #include <errno.h>
4240004   #include <time.h>
4240005   #include <ctype.h>
4240006   //---------------------------------------------------------------------
4240007   static void  usage          (void);
4240008   //---------------------------------------------------------------------
4240009   int
4240010   main (int argc, char *argv[], char *envp[])
4240011   {
4240012       struct tm *timeptr;
4240013       char       string[5];
4240014       time_t     timer;
4240015       int        length;
4240016       char      *input;
4240017       int        i;
4240018       //
4240019       // There can be at most an argument.
4240020       //
4240021       if (argc > 2)
4240022         {
4240023           usage ();
4240024           return (1);
4240025         }
4240026       //
4240027       // Check if there is no argument: must show the date.
```

```
    //
    if (argc == 1)
      {
        timer = time (NULL);
        printf ("%s\n", ctime (&timer));
        return (0);
      }
    //
    // There is one argument and must be the date do set.
    //
    input = argv[1];
    //
    // First get current date, for default values.
    //
    timer   = time (NULL);
    timeptr = gmtime (&timer);
    //
    // Verify to have a correct input.
    //
    length = (int) strlen (input);
    if (length == 8 || length == 10 || length == 12)
      {
        for (i = 0; i < length; i++)
          {
            if (!isdigit (input[i]))
              {
                usage ();
                return (2);
              }
          }
      }
    else
      {
        printf ("input: \"%s\"; length: %i\n", input, length);
        usage ();
        return (3);
      }
    //
    // Select the month.
    //
    string[0] = input[0];
    string[1] = input[1];
    string[2] = '\0';
    timeptr->tm_mon = atoi (string);
    //
    // Select the day.
    //
    string[0] = input[2];
    string[1] = input[3];
    string[2] = '\0';
    timeptr->tm_mday = atoi (string);
    //
    // Select the hour.
    //
    string[0] = input[4];
    string[1] = input[5];
    string[2] = '\0';
    timeptr->tm_hour = atoi (string);
    //
    // Select the minute.
    //
    string[0] = input[6];
    string[1] = input[7];
    string[2] = '\0';
    timeptr->tm_min = atoi (string);
    //
    // Select the year: must verify if there is a century.
    //
    if (length == 12)
      {
        string[0] = input[8];
        string[1] = input[9];
        string[2] = input[10];
        string[3] = input[11];
        string[4] = '\0';
        timeptr->tm_year = atoi (string);
      }
    else if (length == 10)
      {
        sprintf (string, "%04i", timeptr->tm_year);
        string[2] = input[8];
        string[3] = input[9];
        string[4] = '\0';
        timeptr->tm_year = atoi (string);
      }
    //
    // Now convert to 'time_t'.
    //
    timer = mktime (timeptr);
    //
    // Save to the system.
    //
    stime (&timer);
    //
    return (0);
}
//---------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage: date [MMDDHHMM[[CC]YY]]\n");
}
```


applic/ed.c

«

Si veda la sezione u0.9.

```
//---------------------------------------------------------------------
// 2009.08.18
// Modified by Daniele Giacomini for 'os16', to harmonize with it,
// even, when possible, on coding style.
//
// The original was taken form ELKS sources: 'elkscmd/misc_utils/ed.c'.
//---------------------------------------------------------------------
//
// Copyright (c) 1993 by David I. Bell
// Permission is granted to use, distribute, or modify this source,
// provided that this copyright notice remains intact.
//
// The "ed" built-in command (much simplified)
//
//---------------------------------------------------------------------

#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
//---------------------------------------------------------------------
#define isoctal(ch)  (((ch) >= '0') && ((ch) <= '7'))
#define USERSIZE     1024       /* max line length typed in by user */
#define INITBUFSIZE  1024       /* initial buffer size */
//---------------------------------------------------------------------
typedef int num_t;
typedef int len_t;
//
// The following is the type definition of structure 'line_t', but the
// structure contains pointers to the same kind of type. With the
// compiler Bcc, it is the only way to declare it.
//
typedef struct line line_t;
//
struct line {
    line_t *next;
    line_t *prev;
    len_t   len;
    char    data[1];
};
//
static  line_t  lines;
static  line_t *curline;
static  num_t   curnum;
static  num_t   lastnum;
static  num_t   marks[26];
static  bool    dirty;
static  char    *filename;
static  char    searchstring[USERSIZE];
//
static  char   *bufbase;
static  char   *bufptr;
static  len_t   bufused;
static  len_t   bufsize;
//---------------------------------------------------------------------
static  void    docommands  (void);
static  void    subcommand  (char *cp, num_t num1, num_t num2);
static  bool    getnum      (char **retcp, bool *rethavenum,
                             num_t *retnum);
static  bool    setcurnum   (num_t num);
static  bool    initedit    (void);
static  void    termedit    (void);
static  void    addlines    (num_t num);
static  bool    insertline  (num_t num, char *data, len_t len);
static  bool    deletelines (num_t num1, num_t num2);
static  bool    printlines  (num_t num1, num_t num2, bool expandflag);
static  bool    writelines  (char *file, num_t num1, num_t num2);
static  bool    readlines   (char *file, num_t num);
static  num_t   searchlines (char *str, num_t num1, num_t num2);
static  len_t   findstring  (line_t *lp, char *str, len_t len,
                             len_t offset);
static  line_t *findline    (num_t num);
//---------------------------------------------------------------------
// Main.
//---------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    if (!initedit ()) return (2);
    //
    if (argc > 1)
      {
        filename = strdup (argv[1]);
        if (filename == NULL)
          {
            fprintf (stderr, "No memory\n");
            termedit ();
            return (1);
          }
        //
        if (!readlines (filename, 1))
          {
```

```
            termedit ();
            return (0);
          }
        //
        if (lastnum) setcurnum(1);
        //
        dirty = false;
      }
    //
    docommands ();
    //
    termedit ();
    return (0);
}
//---------------------------------------------------------------------
// Read commands until we are told to stop.
//---------------------------------------------------------------------
void
docommands (void)
{
    char   *cp;
    int     len;
    num_t   num1;
    num_t   num2;
    bool    have1;
    bool    have2;
    char    buf[USERSIZE];
    //
    while (true)
      {
        printf(": ");
        fflush (stdout);
        //
        if (fgets (buf, sizeof(buf), stdin) == NULL)
          {
            return;
          }
        //
        len = strlen (buf);
        if (len == 0)
          {
            return;
          }
        //
        cp = &buf[len - 1];
        if (*cp != '\n')
          {
            fprintf(stderr, "Command line too long\n");
            do
              {
                len = fgetc(stdin);
              }
            while ((len != EOF) && (len != '\n'));
            //
            continue;
          }
        //
        while ((cp > buf) && isblank (cp[-1]))
          {
            cp--;
          }
        //
        *cp = '\0';
        //
        cp = buf;
        //
        while (isblank (*cp))
          {
            //*cp++;
            cp++;
          }
        //
        have1 = false;
        have2 = false;
        //
        if ((curnum == 0) && (lastnum > 0))
          {
            curnum = 1;
            curline = lines.next;
          }
        //
        if (!getnum (&cp, &have1, &num1))
          {
            continue;
          }
        //
        while (isblank (*cp))
          {
            cp++;
          }
        //
        if (*cp == ',')
          {
            cp++;
            if (!getnum (&cp, &have2, &num2))
              {
                continue;
              }
            //
            if (!have1)
              {
```



```
                num1 = 1;
              }
            if (!have2)
              {
                num2 = lastnum;
              }
            have1 = true;
            have2 = true;
          }
        //
        if (!have1)
          {
            num1 = curnum;
          }
        if (!have2)
          {
            num2 = num1;
          }
        //
        // Command interpretation switch.
        //
        switch (*cp++)
          {
            case 'a':
                addlines (num1 + 1);
                break;
                //
            case 'c':
                deletelines (num1, num2);
                addlines (num1);
                break;
                //
            case 'd':
                deletelines (num1, num2);
                break;
                //
            case 'f':
                if (*cp && !isblank (*cp))
                  {
                    fprintf (stderr, "Bad file command\n");
                    break;
                  }
                //
                while (isblank (*cp))
                  {
                    cp++;
                  }
                if (*cp == '\0')
                  {
                    if (filename)
                      {
                        printf ("\"%s\"\n", filename);
                      }
                    else
                      {
                        printf ("No filename\n");
                      }
                    break;
                  }
                //
                cp = strdup (cp);
                //
                if (cp == NULL)
                  {
                    fprintf (stderr, "No memory for filename\n");
                    break;
                  }
                //
                if (filename)
                  {
                    free(filename);
                  }
                //
                filename = cp;
                break;
                //
            case 'i':
                addlines (num1);
                break;
                //
            case 'k':
                while (isblank(*cp))
                  {
                    cp++;
                  }
                //
                if ((*cp < 'a') || (*cp > 'a') || cp[1])
                  {
                    fprintf (stderr, "Bad mark name\n");
                    break;
                  }
                //
                marks[*cp - 'a'] = num2;
                break;
                //
            case 'l':
                printlines (num1, num2, true);
                break;
                //
            case 'p':
                printlines (num1, num2, false);
```

```
                break;
                //
            case 'q':
                while (isblank(*cp))
                  {
                    cp++;
                  }
                //
                if (have1 || *cp)
                  {
                    fprintf (stderr, "Bad quit command\n");
                    break;
                  }
                //
                if (!dirty)
                  {
                    return;
                  }
                //
                printf ("Really quit? ");
                fflush (stdout);
                //
                buf[0] = '\0';
                fgets (buf, sizeof(buf), stdin);
                cp = buf;
                //
                while (isblank (*cp))
                  {
                    cp++;
                  }
                //
                if ((*cp == 'y') || (*cp == 'Y'))
                  {
                    return;
                  }
                //
                break;
                //
            case 'r':
                if (*cp && !isblank(*cp))
                  {
                    fprintf (stderr, "Bad read command\n");
                    break;
                  }
                //
                while (isblank(*cp))
                  {
                    cp++;
                  }
                //
                if (*cp == '\0')
                  {
                    fprintf (stderr, "No filename\n");
                    break;
                  }
                //
                if (!have1)
                  {
                    num1 = lastnum;
                  }
                //
                // Open the file and add to the buffer
                // at the next line.
                //
                if (readlines (cp, num1 + 1))
                  {
                    //
                    // If the file open fails, just
                    // break the command.
                    //
                    break;
                  }
                //
                // Set the default file name, if no
                // previous name is available.
                //
                if (filename == NULL)
                  {
                    filename = strdup (cp);
                  }
                //
                break;

            case 's':
                subcommand (cp, num1, num2);
                break;
                //
            case 'w':
                if (*cp && !isblank(*cp))
                  {
                    fprintf(stderr, "Bad write command\n");
                    break;
                  }
                //
                while (isblank(*cp))
                  {
                    cp++;
                  }
                //
                if (!have1)
                  {
```



```
                    num1 = 1;
                    num2 = lastnum;
                  }
                //
                // If the file name is not specified, use the
                // default one.
                //
                if (*cp == '\0')
                  {
                    cp = filename;
                  }
                //
                // If even the default file name is not specified,
                // tell it.
                //
                if (cp == NULL)
                  {
                    fprintf (stderr, "No file name specified\n");
                    break;
                  }
                //
                // Write the file.
                //
                writelines (cp, num1, num2);
                //
                break;
                //
            case 'z':
                switch (*cp)
                  {
                    case '-':
                        printlines (curnum-21, curnum, false);
                        break;
                    case '.':
                        printlines (curnum-11, curnum+10, false);
                        break;
                    default:
                        printlines (curnum, curnum+21, false);
                        break;
                  }
                break;
                //
            case '.':
                if (have1)
                  {
                    fprintf (stderr, "No arguments allowed\n");
                    break;
                  }
                printlines (curnum, curnum, false);
                break;
                //
            case '-':
                if (setcurnum (curnum - 1))
                  {
                    printlines (curnum, curnum, false);
                  }
                break;
                //
            case '=':
                printf ("%d\n", num1);
                break;
                //
            case '\0':
                if (have1)
                  {
                    printlines (num2, num2, false);
                    break;
                  }
                //
                if (setcurnum (curnum + 1))
                  {
                    printlines (curnum, curnum, false);
                  }
                break;
                //
            default:
                fprintf (stderr, "Unimplemented command\n");
                break;
          }
      }
}
//----------------------------------------------------------------------
// Do the substitute command.
// The current line is set to the last substitution done.
//----------------------------------------------------------------------
void
subcommand (char *cp, num_t num1, num_t num2)
{
    int     delim;
    char   *oldstr;
    char   *newstr;
    len_t   oldlen;
    len_t   newlen;
    len_t   deltalen;
    len_t   offset;
    line_t *lp;
    line_t *nlp;
    bool    globalflag;
    bool    printflag;
    bool    didsub;
    bool    needprint;
```

```

    if ((num1 < 1) || (num2 > lastnum) || (num1 > num2))
      {
        fprintf (stderr, "Bad line range for substitute\n");
        return;
      }
    //
    globalflag = false;
    printflag = false;
    didsub = false;
    needprint = false;
    //
    if (isblank (*cp) || (*cp == '\0'))
      {
        fprintf (stderr, "Bad delimiter for substitute\n");
        return;
      }
    //
    delim = *cp++;
    oldstr = cp;
    //
    cp = strchr (cp, delim);
    //
    if (cp == NULL)
      {
        fprintf (stderr, "Missing 2nd delimiter for substitute\n");
        return;
      }
    //
    *cp++ = '\0';
    //
    newstr = cp;
    cp = strchr (cp, delim);
    //
    if (cp)
      {
        *cp++ = '\0';
      }
    else
      {
        cp = "";
      }
    while (*cp)
      {
        switch (*cp++)
          {
            case 'g':
                globalflag = true;
                break;
                //
            case 'p':
                printflag = true;
                break;
                //
            default:
                fprintf (stderr, "Unknown option for substitute\n");
                return;
          }
      }
    //
    if (*oldstr == '\0')
      {
        if (searchstring[0] == '\0')
          {
            fprintf (stderr, "No previous search string\n");
            return;
          }
        oldstr = searchstring;
      }
    //
    if (oldstr != searchstring)
      {
        strcpy (searchstring, oldstr);
      }
    //
    lp = findline (num1);
    if (lp == NULL)
      {
        return;
      }
    //
    oldlen = strlen(oldstr);
    newlen = strlen(newstr);
    deltalen = newlen - oldlen;
    offset = 0;
    //
    while (num1 <= num2)
      {
        offset = findstring (lp, oldstr, oldlen, offset);
        if (offset < 0)
          {
            if (needprint)
              {
                printlines (num1, num1, false);
                needprint = false;
              }
            //
            offset = 0;
            lp = lp->next;
            num1++;
            continue;
```



```
          }
        //
        needprint = printflag;
        didsub = true;
        dirty = true;

        //------------------------------------------------------------
        // If the replacement string is the same size or shorter
        // than the old string, then the substitution is easy.
        //------------------------------------------------------------

        if (deltalen <= 0)
          {
            memcpy (&lp->data[offset], newstr, newlen);
            //
            if (deltalen)
              {
                memcpy (&lp->data[offset + newlen],
                        &lp->data[offset + oldlen],
                        lp->len - offset - oldlen);
                //
                lp->len += deltalen;
              }
            //
            offset += newlen;
            //
            if (globalflag)
              {
                continue;
              }
            //
            if (needprint)
              {
                printlines(num1, num1, false);
                needprint = false;
              }
            //
            lp = nlp->next;
            num1++;
            continue;
          }

        //------------------------------------------------------------
        // The new string is larger, so allocate a new line
        // structure and use that.  Link it in in place of
        // the old line structure.
        //------------------------------------------------------------

        nlp = (line_t *) malloc (sizeof (line_t) + lp->len + deltalen);
        //
        if (nlp == NULL)
          {
            fprintf (stderr, "Cannot get memory for line\n");
            return;
          }
        //
        nlp->len = lp->len + deltalen;
        //
        memcpy (nlp->data, lp->data, offset);
        //
        memcpy (&nlp->data[offset], newstr, newlen);
        //
        memcpy (&nlp->data[offset + newlen],
                &lp->data[offset + oldlen],
                lp->len - offset - oldlen);
        //
        nlp->next = lp->next;
        nlp->prev = lp->prev;
        nlp->prev->next = nlp;
        nlp->next->prev = nlp;
        //
        if (curline == lp)
          {
            curline = nlp;
          }
        //
        free(lp);
        lp = nlp;
        //
        offset += newlen;
        //
        if (globalflag)
          {
            continue;
          }
        //
        if (needprint)
          {
            printlines (num1, num1, false);
            needprint = false;
          }
        //
        lp = lp->next;
        num1++;
      }
    //
    if (!didsub)
      {
        fprintf (stderr, "No substitutions found for \"%s\"\n", oldstr);
      }
}
```

```
4250702 //------------------------------------------------------------------------
4250703 // Search a line for the specified string starting at the specified
4250704 // offset in the line.  Returns the offset of the found string, or -1.
4250705 //------------------------------------------------------------------------
4250706 len_t
4250707 findstring (line_t *lp, char *str, len_t len, len_t offset)
4250708 {
4250709     len_t    left;
4250710     char    *cp;
4250711     char    *ncp;
4250712     //
4250713     cp = &lp->data[offset];
4250714     left = lp->len - offset;
4250715     //
4250716     while (left >= len)
4250717       {
4250718         ncp = memchr(cp, *str, left);
4250719         if (ncp == NULL)
4250720           {
4250721             return (len_t) -1;
4250722           }
4250723         //
4250724         left -= (ncp - cp);
4250725         if (left < len)
4250726           {
4250727             return (len_t) -1;
4250728           }
4250729         //
4250730         cp = ncp;
4250731         if (memcmp(cp, str, len) == 0)
4250732           {
4250733             return (len_t) (cp - lp->data);
4250734           }
4250735         //
4250736         cp++;
4250737         left--;
4250738       }
4250739     //
4250740     return (len_t) -1;
4250741 }
4250742 //------------------------------------------------------------------------
4250743 // Add lines which are typed in by the user.
4250744 // The lines are inserted just before the specified line number.
4250745 // The lines are terminated by a line containing a single dot (ugly!),
4250746 // or by an end of file.
4250747 //------------------------------------------------------------------------
4250748 void
4250749 addlines (num_t num)
4250750 {
4250751     int     len;
4250752     char    buf[USERSIZE + 1];
4250753     //
4250754     while (fgets (buf, sizeof (buf), stdin))
4250755       {
4250756         if ((buf[0] == '.') && (buf[1] == '\n') && (buf[2] == '\0'))
4250757           {
4250758             return;
4250759           }
4250760         //
4250761         len = strlen (buf);
4250762         //
4250763         if (len == 0)
4250764           {
4250765             return;
4250766           }
4250767         //
4250768         if (buf[len - 1] != '\n')
4250769          {
4250770             fprintf (stderr, "Line too long\n");
4250771             //
4250772             do
4250773               {
4250774                 len = fgetc(stdin);
4250775               }
4250776             while ((len != EOF) && (len != '\n'));
4250777             //
4250778             return;
4250779           }
4250780         //
4250781         if (!insertline (num++, buf, len))
4250782           {
4250783             return;
4250784           }
4250785       }
4250786 }
4250787 //------------------------------------------------------------------------
4250788 // Parse a line number argument if it is present.  This is a sum
4250789 // or difference of numbers, '.', '$', 'x, or a search string.
4250790 // Returns true if successful (whether or not there was a number).
4250791 // Returns false if there was a parsing error, with a message output.
4250792 // Whether there was a number is returned indirectly, as is the number.
4250793 // The character pointer which stopped the scan is also returned.
4250794 //------------------------------------------------------------------------
4250795 static bool
4250796 getnum (char **retcp, bool *rethavenum, num_t *retnum)
4250797 {
4250798     char   *cp;
4250799     char   *str;
4250800     bool    havenum;
4250801     num_t   value;
4250802     num_t   num;
```



```
4250803     num_t   sign;
4250804     //
4250805     cp = *retcp;
4250806     havenum = false;
4250807     value = 0;
4250808     sign = 1;
4250809     //
4250810     while (true)
4250811       {
4250812         while (isblank(*cp))
4250813           {
4250814             cp++;
4250815           }
4250816         //
4250817         switch (*cp)
4250818           {
4250819             case '.':
4250820                 havenum = true;
4250821                 num = curnum;
4250822                 cp++;
4250823                 break;
4250824                 //
4250825             case '$':
4250826                 havenum = true;
4250827                 num = lastnum;
4250828                 cp++;
4250829                 break;
4250830                 //
4250831             case '\'':
4250832                 cp++;
4250833                 if ((*cp < 'a') || (*cp > 'z'))
4250834                   {
4250835                     fprintf (stderr, "Bad mark name\n");
4250836                     return false;
4250837                   }
4250838                 //
4250839                 havenum = true;
4250840                 num = marks[*cp++ - 'a'];
4250841                 break;
4250842                 //
4250843             case '/':
4250844                 str = ++cp;
4250845                 cp = strchr (str, '/');
4250846                 if (cp)
4250847                   {
4250848                     *cp++ = '\0';
4250849                   }
4250850                 else
4250851                   {
4250852                     cp = "";
4250853                   }
4250854                 num = searchlines (str, curnum, lastnum);
4250855                 if (num == 0)
4250856                   {
4250857                     return false;
4250858                   }
4250859                 //
4250860                 havenum = true;
4250861                 break;
4250862                 //
4250863             default:
4250864                 if (!isdigit (*cp))
4250865                  {
4250866                     *retcp = cp;
4250867                     *rethavenum = havenum;
4250868                     *retnum = value;
4250869                     return true;
4250870                   }
4250871                 //
4250872                 num = 0;
4250873                 while (isdigit(*cp))
4250874                   {
4250875                     num = num * 10 + *cp++ - '0';
4250876                   }
4250877                 havenum = true;
4250878                 break;
4250879           }
4250880         //
4250881         value += num * sign;
4250882         //
4250883         while (isblank (*cp))
4250884           {
4250885             cp++;
4250886           }
4250887         //
4250888         switch (*cp)
4250889           {
4250890             case '-':
4250891                 sign = -1;
4250892                 cp++;
4250893                 break;
4250894                 //
4250895             case '+':
4250896                 sign = 1;
4250897                 cp++;
4250898                 break;
4250899                 //
4250900             default:
4250901                 *retcp = cp;
4250902                 *rethavenum = havenum;
4250903                 *retnum = value;
```

```
                return true;
            }
        }
}
//---------------------------------------------------------------------
// Initialize everything for editing.
//---------------------------------------------------------------------
bool
initedit (void)
{
    int i;
    //
    bufsize = INITBUFSIZE;
    bufbase = malloc (bufsize);
    //
    if (bufbase == NULL)
      {
        fprintf (stderr, "No memory for buffer\n");
        return false;
      }
    //
    bufptr = bufbase;
    bufused = 0;
    //
    lines.next = &lines;
    lines.prev = &lines;
    //
    curline = NULL;
    curnum = 0;
    lastnum = 0;
    dirty = false;
    filename = NULL;
    searchstring[0] = '\0';
    //
    for (i = 0; i < 26; i++)
      {
        marks[i] = 0;
      }
    //
    return true;
}
//---------------------------------------------------------------------
// Finish editing.
//---------------------------------------------------------------------
void
termedit (void)
{
    if (bufbase) free(bufbase);
    bufbase = NULL;
    //
    bufptr = NULL;
    bufsize = 0;
    bufused = 0;
    //
    if (filename) free(filename);
    filename = NULL;
    //
    searchstring[0] = '\0';
    //
    if (lastnum) deletelines (1, lastnum);
    //
    lastnum = 0;
    curnum = 0;
    curline = NULL;
}
//---------------------------------------------------------------------
// Read lines from a file at the specified line number.
// Returns true if the file was successfully read.
//---------------------------------------------------------------------
bool
readlines (char *file, num_t num)
{
    int     fd;
    int     cc;
    len_t   len;
    len_t   linecount;
    len_t   charcount;
    char   *cp;
    //
    if ((num < 1) || (num > lastnum + 1))
      {
        fprintf (stderr, "Bad line for read\n");
        return false;
      }
    //
    fd = open (file, O_RDONLY);
    if (fd < 0)
      {
        perror (file);
        return false;
      }
    //
    bufptr = bufbase;
    bufused = 0;
    linecount = 0;
    charcount = 0;
    //
    printf ("\"%s\", ", file);
    fflush(stdout);
    //
    do
```



```
      {
        cp = memchr(bufptr, '\n', bufused);
        if (cp)
          {
            len = (cp - bufptr) + 1;
            //
            if (!insertline (num, bufptr, len))
              {
                close (fd);
                return false;
              }
            //
            bufptr += len;
            bufused -= len;
            charcount += len;
            linecount++;
            num++;
            continue;
          }
        //
        if (bufptr != bufbase)
          {
            memcpy (bufbase, bufptr, bufused);
            bufptr = bufbase + bufused;
          }
        //
        if (bufused >= bufsize)
          {
            len = (bufsize * 3) / 2;
            cp = realloc (bufbase, len);
            if (cp == NULL)
              {
                fprintf (stderr, "No memory for buffer\n");
                close (fd);
                return false;
              }
            //
            bufbase = cp;
            bufptr = bufbase + bufused;
            bufsize = len;
          }
        //
        cc = read (fd, bufptr, bufsize - bufused);
        bufused += cc;
        bufptr = bufbase;
      }
    while (cc > 0);
    //
    if (cc < 0)
      {
        perror (file);
        close (fd);
        return false;
      }
    //
    if (bufused)
      {
        if (!insertline (num, bufptr, bufused))
          {
            close (fd);
            return -1;
          }
        linecount++;
        charcount += bufused;
      }
    //
    close (fd);
    //
    printf ("%d lines%s, %d chars\n",
            linecount,
            (bufused ? " (incomplete)" : ""),
            charcount);
    //
    return true;
}
//---------------------------------------------------------------------
// Write the specified lines out to the specified file.
// Returns true if successful, or false on an error with a message
// output.
//---------------------------------------------------------------------
bool
writelines (char *file, num_t num1, num_t num2)
{
    int     fd;
    line_t *lp;
    len_t   linecount;
    len_t   charcount;
    //
    if ((num1 < 1) || (num2 > lastnum) || (num1 > num2))
      {
        fprintf (stderr, "Bad line range for write\n");
        return false;
      }
    //
    linecount = 0;
    charcount = 0;
    //
    fd = creat (file, 0666);
    if (fd < 0)
      {
        perror (file);
```

```
          return false;
        }
      //
      printf("\"%s\", ", file);
      fflush (stdout);
      //
      lp = findline (num1);
      if (lp == NULL)
        {
          close (fd);
          return false;
        }
      //
      while (num1++ <= num2)
        {
          if (write(fd, lp->data, lp->len) != lp->len)
            {
              perror(file);
              close(fd);
              return false;
            }
          //
          charcount += lp->len;
          linecount++;
          lp = lp->next;
        }
      //
      if (close(fd) < 0)
        {
          perror(file);
          return false;
        }
      //
      printf ("%d lines, %d chars\n", linecount, charcount);
      //
      return true;
}
//---------------------------------------------------------------------
// Print lines in a specified range.
// The last line printed becomes the current line.
// If expandflag is true, then the line is printed specially to
// show magic characters.
//---------------------------------------------------------------------
bool
printlines (num_t num1, num_t num2, bool expandflag)
{
    line_t        *lp;
    unsigned char *cp;
    int            ch;
    len_t          count;
    //
    if ((num1 < 1) || (num2 > lastnum) || (num1 > num2))
      {
        fprintf (stderr, "Bad line range for print\n");
        return false;
      }
    //
    lp = findline (num1);
    if (lp == NULL)
      {
        return false;
      }
    //
    while (num1 <= num2)
      {
        if (!expandflag)
          {
            write (STDOUT_FILENO, lp->data, lp->len);
            setcurnum (num1++);
            lp = lp->next;
            continue;
          }

        //-------------------------------------------------------------
        // Show control characters and characters with the
        // high bit set specially.
        //-------------------------------------------------------------

        cp = (unsigned char *) lp->data;
        count = lp->len;
        //
        if ((count > 0) && (cp[count - 1] == '\n'))
          {
            count--;
          }
        //
        while (count-- > 0)
          {
            ch = *cp++;
            if (ch & 0x80)
              {
                fputs ("M-", stdout);
                ch &= 0x7f;
              }
            if (ch < ' ')
              {
                fputc ('^', stdout);
                ch += '@';
              }
            if (ch == 0x7f)
              {
```



```
                fputc ('^', stdout);
                ch = '?';
              }
            fputc (ch, stdout);
          }
        //
        fputs ("$\n", stdout);
        //
        setcurnum (num1++);
        lp = lp->next;
      }
    //
    return true;
}
//---------------------------------------------------------------------
// Insert a new line with the specified text.
// The line is inserted so as to become the specified line,
// thus pushing any existing and further lines down one.
// The inserted line is also set to become the current line.
// Returns true if successful.
//---------------------------------------------------------------------
bool
insertline (num_t num, char *data, len_t len)
{
    line_t    *newlp;
    line_t    *lp;
    //
    if ((num < 1) || (num > lastnum + 1))
      {
        fprintf (stderr, "Inserting at bad line number\n");
        return false;
      }
    //
    newlp = (line_t *) malloc (sizeof (line_t) + len - 1);
    if (newlp == NULL)
      {
        fprintf (stderr, "Failed to allocate memory for line\n");
        return false;
      }
    //
    memcpy (newlp->data, data, len);
    newlp->len = len;
    //
    if (num > lastnum)
      {
        lp = &lines;
      }
    else
      {
        lp = findline (num);
        if (lp == NULL)
          {
            free ((char *) newlp);
            return false;
          }
      }
    //
    newlp->next = lp;
    newlp->prev = lp->prev;
    lp->prev->next = newlp;
    lp->prev = newlp;
    //
    lastnum++;
    dirty = true;
    //
    return setcurnum (num);
}
//---------------------------------------------------------------------
// Delete lines from the given range.
//---------------------------------------------------------------------
bool
deletelines (num_t num1, num_t num2)
{
    line_t   *lp;
    line_t   *nlp;
    line_t   *plp;
    num_t     count;
    //
    if ((num1 < 1) || (num2 > lastnum) || (num1 > num2))
      {
        fprintf (stderr, "Bad line numbers for delete\n");
        return false;
      }
    //
    lp = findline (num1);
    if (lp == NULL)
      {
        return false;
      }
    //
    if ((curnum >= num1) && (curnum <= num2))
      {
        if (num2 < lastnum)
          {
            setcurnum (num2 + 1);
          }
        else if (num1 > 1)
          {
            setcurnum (num1 - 1);
          }
        else
```

```
4251308           {
4251309             curnum = 0;
4251310           }
4251311       }
4251312     //
4251313     count = num2 - num1 + 1;
4251314     //
4251315     if (curnum > num2)
4251316       {
4251317         curnum -= count;
4251318       }
4251319     //
4251320     lastnum -= count;
4251321     //
4251322     while (count-- > 0)
4251323       {
4251324         nlp = lp->next;
4251325         plp = lp->prev;
4251326         plp->next = nlp;
4251327         nlp->prev = plp;
4251328         lp->next = NULL;
4251329         lp->prev = NULL;
4251330         lp->len = 0;
4251331         free(lp);
4251332         lp = nlp;
4251333       }
4251334     //
4251335     dirty = true;
4251336     //
4251337     return true;
4251338 }
4251339 //------------------------------------------------------------------------
4251340 // Search for a line which contains the specified string.
4251341 // If the string is NULL, then the previously searched for string
4251342 // is used.  The currently searched for string is saved for future use.
4251343 // Returns the line number which matches, or 0 if there was no match
4251344 // with an error printed.
4251345 //------------------------------------------------------------------------
4251346 num_t
4251347 searchlines (char *str, num_t num1, num_t num2)
4251348 {
4251349     line_t *lp;
4251350     int     len;
4251351     //
4251352     if ((num1 < 1) || (num2 > lastnum) || (num1 > num2))
4251353       {
4251354         fprintf (stderr, "Bad line numbers for search\n");
4251355         return 0;
4251356       }
4251357     //
4251358     if (*str == '\0')
4251359       {
4251360         if (searchstring[0] == '\0')
4251361           {
4251362             fprintf(stderr, "No previous search string\n");
4251363             return 0;
4251364           }
4251365         str = searchstring;
4251366       }
4251367     //
4251368     if (str != searchstring)
4251369       {
4251370         strcpy(searchstring, str);
4251371       }
4251372     //
4251373     len = strlen(str);
4251374     //
4251375     lp = findline (num1);
4251376     if (lp == NULL)
4251377       {
4251378         return 0;
4251379       }
4251380     //
4251381     while (num1 <= num2)
4251382       {
4251383         if (findstring(lp, str, len, 0) >= 0)
4251384           {
4251385             return num1;
4251386           }
4251387         //
4251388         num1++;
4251389         lp = lp->next;
4251390       }
4251391     //
4251392     fprintf (stderr, "Cannot find string \"%s\"\n", str);
4251393     //
4251394     return 0;
4251395 }
4251396 //------------------------------------------------------------------------
4251397 // Return a pointer to the specified line number.
4251398 //------------------------------------------------------------------------
4251399 line_t *
4251400 findline (num_t num)
4251401 {
4251402     line_t   *lp;
4251403     num_t     lnum;
4251404     //
4251405     if ((num < 1) || (num > lastnum))
4251406       {
4251407         fprintf (stderr, "Line number %d does not exist\n", num);
4251408         return NULL;
```



```
4251409       }
4251410     //
4251411     if (curnum <= 0)
4251412       {
4251413         curnum = 1;
4251414         curline = lines.next;
4251415       }
4251416     //
4251417     if (num == curnum)
4251418       {
4251419         return curline;
4251420       }
4251421     //
4251422     lp = curline;
4251423     lnum = curnum;
4251424     //
4251425     if (num < (curnum / 2))
4251426       {
4251427         lp = lines.next;
4251428         lnum = 1;
4251429       }
4251430     else if (num > ((curnum + lastnum) / 2))
4251431       {
4251432         lp = lines.prev;
4251433         lnum = lastnum;
4251434       }
4251435     //
4251436     while (lnum < num)
4251437       {
4251438         lp = lp->next;
4251439         lnum++;
4251440       }
4251441     //
4251442     while (lnum > num)
4251443       {
4251444         lp = lp->prev;
4251445         lnum--;
4251446       }
4251447     //
4251448     return lp;
4251449 }
4251450 //------------------------------------------------------------------------
4251451 // Set the current line number.
4251452 // Returns true if successful.
4251453 //------------------------------------------------------------------------
4251454 bool
4251455 setcurnum (num_t num)
4251456 {
4251457     line_t    *lp;
4251458     //
4251459     lp = findline (num);
4251460     if (lp == NULL)
4251461       {
4251462         return false;
4251463       }
4251464     //
4251465     curnum = num;
4251466     curline = lp;
4251467     //
4251468     return true;
4251469 }
4251470
4251471 /* END CODE */
```

applic/getty.c

Si veda la sezione u0.1.

```
4260001 #include <unistd.h>
4260002 #include <stdio.h>
4260003 #include <stdlib.h>
4260004 #include <signal.h>
4260005 #include <sys/wait.h>
4260006 #include <limits.h>
4260007 #include <sys/os16.h>
4260008 #include <fcntl.h>
4260009 #include <stdio.h>
4260010 //------------------------------------------------------------------------
4260011 int
4260012 main (int argc, char *argv[], char *envp[])
4260013 {
4260014     char   *device_name;
4260015     int     fdn;
4260016     char   *exec_argv[2];
4260017     char  **exec_envp;
4260018     char    buffer[BUFSIZ];
4260019     ssize_t size_read;
4260020     int     status;
4260021     //
4260022     // The first argument is mandatory and must be a console terminal.
4260023     //
4260024     device_name = argv[1];
4260025     //
4260026     // A console terminal is correctly selected (but it is not checked
4260027     // if it is a really available one).
4260028     // Set as a process group leader.
4260029     //
4260030     setpgrp ();
4260031     //
4260032     // Open the terminal, that should become the controlling terminal:
4260033     // close the standard input and open the new terminal (r/w).
```

```
4260034     //
4260035     close (0);
4260036     fdn = open (device_name, O_RDWR);
4260037     if (fdn < 0)
4260038       {
4260039         //
4260040         // Cannot open terminal. A message should appear, at least
4260041         // to the current console.
4260042         //
4260043         perror (NULL);
4260044         return (-1);
4260045       }
4260046     //
4260047     // Reset terminal device permissions and ownership.
4260048     //
4260049     status = fchown (fdn, (uid_t) 0, (gid_t) 0);
4260050     if (status != 0)
4260051       {
4260052         perror (NULL);
4260053       }
4260054     status = fchmod (fdn, 0644);
4260055     if (status != 0)
4260056       {
4260057         perror (NULL);
4260058       }
4260059     //
4260060     // The terminal is open and it should be already the controlling
4260061     // one: show '/etc/issue'. The same variable 'fdn' is used, because
4260062     // the controlling terminal will never be closed (the exit syscall
4260063     // will do it).
4260064     //
4260065     fdn = open ("/etc/issue", O_RDONLY);
4260066     if (fdn > 0)
4260067       {
4260068         //
4260069         // The file is present and is shown.
4260070         //
4260071         for (size_read = 1; size_read > 0;)
4260072           {
4260073             size_read = read (fdn, buffer, (size_t) (BUFSIZ - 1));
4260074             if (size_read < 0)
4260075               {
4260076                 break;
4260077               }
4260078             buffer[size_read] = '\0';
4260079             printf ("%s", buffer);
4260080           }
4260081         close (fdn);
4260082       }
4260083     //
4260084     // Show the terminal.
4260085     //
4260086     printf ("This is terminal %s\n", device_name);
4260087     //
4260088     // It is time to exec login: the environment is inherited directly
4260089     // from 'init'.
4260090     //
4260091     exec_argv[0] = "login";
4260092     exec_argv[1] = NULL;
4260093     exec_envp    = envp;
4260094     execve ("/bin/login", exec_argv, exec_envp);
4260095     //
4260096     // If 'execve()' returns, it is an error.
4260097     //
4260098     exit (-1);
4260099 }
```

applic/init.c

«

Si veda la sezione u0.2.

```
4270001 #include <unistd.h>
4270002 #include <stdio.h>
4270003 #include <stdlib.h>
4270004 #include <signal.h>
4270005 #include <sys/wait.h>
4270006 #include <limits.h>
4270007 #include <sys/os16.h>
4270008 #include <fcntl.h>
4270009 #include <string.h>
4270010 //---------------------------------------------------------------------
4270011 #define RESPAWN_MAX        7
4270012 #define COMMAND_MAX      100
4270013 #define LINE_MAX        1024
4270014 //---------------------------------------------------------------------
4270015 int
4270016 main (int argc, char *argv[], char *envp[])
4270017 {
4270018     //
4270019     // 'init.c' has its own 'init.crt0.s' with a very small stack
4270020     // size. Remember to verify to have enough room for the stack.
4270021     //
4270022     pid_t  pid;
4270023     int    status;
4270024     char  *exec_argv[3];
4270025     char  *exec_envp[3];
4270026     char   buffer[LINE_MAX];
4270027     int    r;                   // Respawn table index.
4270028     int    b;                   // Buffer index.
4270029     size_t size_read;
4270030     char  *inittab_id;
```



```
4270031     char  *inittab_runlevels;
4270032     char  *inittab_action;
4270033     char  *inittab_process;
4270034     int    eof;
4270035     int    fd;
4270036     //
4270037     // It follows a table for commands to be respawn.
4270038     //
4270039     struct {
4270040         pid_t   pid;
4270041         char    command[COMMAND_MAX];
4270042     } respawn[RESPAWN_MAX];
4270043
4270044     //---------------------------------------------------------------
4270045     signal (SIGHUP,  SIG_IGN);
4270046     signal (SIGINT,  SIG_IGN);
4270047     signal (SIGQUIT, SIG_IGN);
4270048     signal (SIGILL,  SIG_IGN);
4270049     signal (SIGABRT, SIG_IGN);
4270050     signal (SIGFPE,  SIG_IGN);
4270051 //  signal (SIGKILL, SIG_IGN);  Cannot ignore SIGKILL.
4270052     signal (SIGSEGV, SIG_IGN);
4270053     signal (SIGPIPE, SIG_IGN);
4270054     signal (SIGALRM, SIG_IGN);
4270055     signal (SIGTERM, SIG_IGN);
4270056 //  signal (SIGSTOP, SIG_IGN);  Cannot ignore SIGSTOP.
4270057     signal (SIGTSTP, SIG_IGN);
4270058     signal (SIGCONT, SIG_IGN);
4270059     signal (SIGTTIN, SIG_IGN);
4270060     signal (SIGTTOU, SIG_IGN);
4270061     signal (SIGUSR1, SIG_IGN);
4270062     signal (SIGUSR2, SIG_IGN);
4270063     //---------------------------------------------------------------
4270064     printf ("init\n");
4270065     //  heap_clear ();
4270066     //  process_info ();
4270067     //---------------------------------------------------------------
4270068     //
4270069     // Reset the 'respawn' table.
4270070     //
4270071     for (r = 0; r < RESPAWN_MAX; r++)
4270072       {
4270073         respawn[r].pid = 0;
4270074         respawn[r].command[0]                  = 0;
4270075         respawn[r].command[COMMAND_MAX-1] = 0;
4270076       }
4270077     //
4270078     // Read the '/etc/inittab' file.
4270079     //
4270080     fd = open ("/etc/inittab", O_RDONLY);
4270081     //
4270082     if (fd < 0)
4270083       {
4270084         perror ("Cannot open file '/etc/inittab'");
4270085         exit (-1);
4270086       }
4270087     //
4270088     //
4270089     //
4270090     for (eof = 0, r = 0; !eof && r < RESPAWN_MAX; r++)
4270091       {
4270092         for (b = 0; b < LINE_MAX; b++)
4270093           {
4270094             size_read = read (fd, &buffer[b], (size_t) 1);
4270095             if (size_read <= 0)
4270096               {
4270097                 buffer[b] = 0;
4270098                 eof       = 1;           // Close the read loop.
4270099                 break;
4270100               }
4270101             if (buffer[b] == '\n')
4270102               {
4270103                 buffer[b]  = 0;
4270104                 break;
4270105               }
4270106           }
4270107         //
4270108         // Remove comments: just replace '#' with '\0'.
4270109         //
4270110         for (b = 0; b < LINE_MAX; b++)
4270111           {
4270112             if (buffer[b] == '#')
4270113               {
4270114                 buffer[b] = 0;
4270115                 break;
4270116               }
4270117           }
4270118         //
4270119         // If the buffer is an empty string, just loop to next
4270120         // record.
4270121         //
4270122         if (strlen (buffer) == 0)
4270123           {
4270124             r--;
4270125             continue;
4270126           }
4270127         //
4270128         //
4270129         //
4270130         inittab_id        = strtok (buffer, ":");
4270131         inittab_runlevels = strtok (NULL, ":");
```

```
4270132         inittab_action    = strtok (NULL, ":");
4270133         inittab_process   = strtok (NULL, ":");
4270134         //
4270135         // Only action 'respawn' is used.
4270136         //
4270137         if (strcmp (inittab_action, "respawn") == 0)
4270138           {
4270139             strncpy (respawn[r].command, inittab_process, COMMAND_MAX);
4270140           }
4270141         else
4270142           {
4270143             r--;
4270144           }
4270145       }
4270146     //
4270147     //
4270148     //
4270149     close (fd);
4270150     //
4270151     // Define common environment.
4270152     //
4270153     exec_envp[0] = "PATH=/bin:/usr/bin:/sbin:/usr/sbin";
4270154     exec_envp[1] = "CONSOLE=/dev/console";
4270155     exec_envp[2] = NULL;
4270156     //
4270157     // Start processes.
4270158     //
4270159     for (r = 0; r < RESPAWN_MAX; r++)
4270160       {
4270161         if (strlen (respawn[r].command) > 0)
4270162           {
4270163             respawn[r].pid = fork ();
4270164             if (respawn[r].pid == 0)
4270165               {
4270166                 exec_argv[0] = strtok (respawn[r].command, " \t");
4270167                 exec_argv[1] = strtok (NULL, " \t");
4270168                 exec_argv[2] = NULL;
4270169                 execve (exec_argv[0], exec_argv, exec_envp);
4270170                 perror (NULL);
4270171                 exit (0);
4270172               }
4270173           }
4270174       }
4270175     //
4270176     // Wait for the death of child.
4270177     //
4270178     while (1)
4270179       {
4270180         pid = wait (&status);
4270181         for (r = 0; r < RESPAWN_MAX; r++)
4270182           {
4270183             if (pid == respawn[r].pid)
4270184               {
4270185                 //
4270186                 // Run it again.
4270187                 //
4270188                 respawn[r].pid = fork ();
4270189                 if (respawn[r].pid == 0)
4270190                   {
4270191                     exec_argv[0] = strtok (respawn[r].command, " \t");
4270192                     exec_argv[1] = strtok (NULL, " \t");
4270193                     exec_argv[2] = NULL;
4270194                     execve (exec_argv[0], exec_argv, exec_envp);
4270195                     exit (0);
4270196                   }
4270197                 break;
4270198               }
4270199           }
4270200       }
4270201 }
```

applic/kill.c

«

Si veda la sezione u0.10.

```
4280001 #include <sys/os16.h>
4280002 #include <sys/stat.h>
4280003 #include <sys/types.h>
4280004 #include <unistd.h>
4280005 #include <stdlib.h>
4280006 #include <fcntl.h>
4280007 #include <errno.h>
4280008 #include <signal.h>
4280009 #include <stdio.h>
4280010 #include <string.h>
4280011 #include <limits.h>
4280012 #include <libgen.h>
4280013 //------------------------------------------------------------------------
4280014 static void usage         (void);
4280015 //------------------------------------------------------------------------
4280016 int
4280017 main (int argc, char *argv[], char *envp[])
4280018 {
4280019     int          signal;
4280020     int          pid;
4280021     int          a;             // Index inside arguments.
4280022     int          option_s = 0;
4280023     int          option_l = 0;
4280024     int          opt;
4280025     extern char *optarg;
4280026     extern int   optopt;
```



```
4280027     //
4280028     // There must be at least an option, plus the program name.
4280029     //
4280030     if (argc < 2)
4280031       {
4280032         usage ();
4280033         return (1);
4280034       }
4280035     //
4280036     // Check for options.
4280037     //
4280038     while ((opt = getopt (argc, argv, ":ls:")) != -1)
4280039       {
4280040         switch (opt)
4280041           {
4280042             case 'l':
4280043                 option_l = 1;
4280044                 break;
4280045             case 's':
4280046                 option_s = 1;
4280047                 //
4280048                 // In that case, there must be at least three arguments:
4280049                 // the option, the signal and the process id.
4280050                 //
4280051                 if (argc < 4)
4280052                   {
4280053                     usage ();
4280054                     return (1);
4280055                   }
4280056                 //
4280057                 // Argument numbers are ok. Check the signal.
4280058                 //
4280059                 if (strcmp (optarg, "HUP") == 0)
4280060                   {
4280061                     signal = SIGHUP;
4280062                   }
4280063                 else if (strcmp (optarg, "INT") == 0)
4280064                   {
4280065                     signal = SIGINT;
4280066                   }
4280067                 else if (strcmp (optarg, "QUIT") == 0)
4280068                   {
4280069                     signal = SIGQUIT;
4280070                   }
4280071                 else if (strcmp (optarg, "ILL") == 0)
4280072                   {
4280073                     signal = SIGILL;
4280074                   }
4280075                 else if (strcmp (optarg, "ABRT") == 0)
4280076                   {
4280077                     signal = SIGABRT;
4280078                   }
4280079                 else if (strcmp (optarg, "FPE") == 0)
4280080                   {
4280081                     signal = SIGFPE;
4280082                   }
4280083                 else if (strcmp (optarg, "KILL") == 0)
4280084                   {
4280085                     signal = SIGKILL;
4280086                   }
4280087                 else if (strcmp (optarg, "SEGV") == 0)
4280088                   {
4280089                     signal = SIGSEGV;
4280090                   }
4280091                 else if (strcmp (optarg, "PIPE") == 0)
4280092                   {
4280093                     signal = SIGPIPE;
4280094                   }
4280095                 else if (strcmp (optarg, "ALRM") == 0)
4280096                   {
4280097                     signal = SIGALRM;
4280098                   }
4280099                 else if (strcmp (optarg, "TERM") == 0)
4280100                   {
4280101                     signal = SIGTERM;
4280102                   }
4280103                 else if (strcmp (optarg, "STOP") == 0)
4280104                   {
4280105                     signal = SIGSTOP;
4280106                   }
4280107                 else if (strcmp (optarg, "TSTP") == 0)
4280108                   {
4280109                     signal = SIGTSTP;
4280110                   }
4280111                 else if (strcmp (optarg, "CONT") == 0)
4280112                   {
4280113                     signal = SIGCONT;
4280114                   }
4280115                 else if (strcmp (optarg, "CHLD") == 0)
4280116                   {
4280117                     signal = SIGCHLD;
4280118                   }
4280119                 else if (strcmp (optarg, "TTIN") == 0)
4280120                   {
4280121                     signal = SIGTTIN;
4280122                   }
4280123                 else if (strcmp (optarg, "TTOU") == 0)
4280124                   {
4280125                     signal = SIGTTOU;
4280126                   }
4280127                 else if (strcmp (optarg, "USR1") == 0)
```

```
                  {
                    signal = SIGUSR1;
                  }
                else if (strcmp (optarg, "USR2") == 0)
                  {
                    signal = SIGUSR2;
                  }
                else
                  {
                    fprintf (stderr, "Unknown signal %s.\n", optarg);
                    return (1);
                  }
                break;
            case '?':
                fprintf (stderr, "Unknown option -%c.\n", optopt);
                usage ();
                return (1);
                break;
            case ':':
                fprintf (stderr, "Missing argument for option -%c\n",
                         optopt);
                usage ();
                return (1);
                break;
            default:
                fprintf (stderr, "Getopt problem: unknown option %c\n",
                         opt);
                return (1);
            }
        }
      //
      //
      //
      if (option_l && option_s)
        {
          fprintf (stderr, "Options \"-l\" and \"-s\" together ");
          fprintf (stderr, "are incompatible.\n");
          usage ();
          return (1);
        }
      //
      // Option "-l".
      //
      if (option_l)
        {
          printf ("HUP ");
          printf ("INT ");
          printf ("QUIT ");
          printf ("ILL ");
          printf ("ABRT ");
          printf ("FPE ");
          printf ("KILL ");
          printf ("SEGV ");
          printf ("PIPE ");
          printf ("ALRM ");
          printf ("TERM ");
          printf ("STOP ");
          printf ("TSTP ");
          printf ("CONT ");
          printf ("CHLD ");
          printf ("TTIN ");
          printf ("TTOU ");
          printf ("USR1 ");
          printf ("USR2 ");
          printf ("\n");
        }
      //
      // Option "-s".
      //
      if (option_s)
        {
          //
          // Scan arguments.
          //
          for (a = 3; a < argc; a++)
            {
              //
              // Get PID.
              //
              pid = atoi (argv[a]);
              if (pid > 0)
                {
                  //
                  // Kill.
                  //
                  if (kill (pid, signal) < 0)
                    {
                      perror (argv[a]);
                    }
                }
              else
                {
                  fprintf (stderr, "Invalid PID %s.", argv[a]);
                }
            }
        }
      //
      // All done.
      //
      return (0);
}
```



```
//------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage: kill -s SIGNAL_NAME PID...\n");
    fprintf (stderr, "       kill -l\n");
}
```

applic/ln.c

«

Si veda la sezione u0.11.

```
#include <sys/os16.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
//------------------------------------------------------------------
static void usage (void);
//------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    char       *source;
    char       *destination;
    char       *new_destination;
    struct stat file_status;
    int         dest_is_a_dir = 0;
    int         a;                      // Argument index.
    char        path[PATH_MAX];
    //
    // There must be at least two arguments, plus the program name.
    //
    if (argc < 3)
      {
        usage ();
        return (1);
      }
    //
    // Select the last argument as the destination.
    //
    destination = argv[argc-1];
    //
    // Check if it is a directory and save it in a flag.
    //
    if (stat (destination, &file_status) == 0)
      {
        if (S_ISDIR (file_status.st_mode))
          {
            dest_is_a_dir = 1;
          }
      }
    //
    // If there are more than two arguments, verify that the last
    // one is a directory.
    //
    if (argc > 3)
      {
        if (!dest_is_a_dir)
          {
            usage ();
            fprintf (stderr, "The destination \"%s\" ",
                     destination);
            fprintf (stderr, "is not a directory!\n");
            return (1);
          }
      }
    //
    // Scan the arguments, excluded the last, that is the destination.
    //
    for (a = 1; a < (argc - 1); a++)
      {
        //
        // Source.
        //
        source = argv[a];
        //
        // Verify access permissions.
        //
        if (access (source, R_OK) < 0)
          {
            perror (source);
            continue;
          }
        //
        // Destination.
        //
        // If it is a directory, the destination path
        // must be corrected.
        //
        if (dest_is_a_dir)
          {
            path[0] = 0;
            strcat (path, destination);
```

```
4290090             strcat (path, "/");
4290091             strcat (path, basename (source));
4290092             //
4290093             // Update the destination path.
4290094             //
4290095             new_destination = path;
4290096           }
4290097         else
4290098           {
4290099             new_destination = destination;
4290100           }
4290101         //
4290102         // Check if destination file exists.
4290103         //
4290104         if (stat (new_destination, &file_status) == 0)
4290105           {
4290106             fprintf (stderr, "The destination file, \"%s\", ",
4290107                      new_destination);
4290108             fprintf (stderr, "already exists!\n");
4290109             continue;
4290110           }
4290111         //
4290112         // Everything is ready for the link.
4290113         //
4290114         if (link (source, new_destination) < 0)
4290115           {
4290116             perror (new_destination);
4290117             continue;
4290118           }
4290119       }
4290120     //
4290121     // All done.
4290122     //
4290123     return (0);
4290124 }
4290125 //---------------------------------------------------------------------------
4290126 static void
4290127 usage (void)
4290128 {
4290129     fprintf (stderr, "Usage: ln OLD_NAME NEW_NAME\n");
4290130     fprintf (stderr, "       ln FILE... DIRECTORY\n");
4290131 }
```

applic/login.c

«

Si veda la sezione u0.12.

```
4300001 #include <unistd.h>
4300002 #include <stdlib.h>
4300003 #include <sys/stat.h>
4300004 #include <sys/types.h>
4300005 #include <fcntl.h>
4300006 #include <errno.h>
4300007 #include <unistd.h>
4300008 #include <signal.h>
4300009 #include <stdio.h>
4300010 #include <sys/wait.h>
4300011 #include <stdio.h>
4300012 #include <string.h>
4300013 #include <limits.h>
4300014 #include <stdint.h>
4300015 #include <sys/os16.h>
4300016 //---------------------------------------------------------------------------
4300017 #define LOGIN_MAX       64
4300018 #define PASSWORD_MAX    64
4300019 #define HOME_MAX        64
4300020 #define LINE_MAX        1024
4300021 //---------------------------------------------------------------------------
4300022 int
4300023 main (int argc, char *argv[], char *envp[])
4300024 {
4300025     char    login[LOGIN_MAX];
4300026     char    password[PASSWORD_MAX];
4300027     char    buffer[LINE_MAX];
4300028     char   *user_name;
4300029     char   *user_password;
4300030     char   *user_uid;
4300031     char   *user_gid;
4300032     char   *user_description;
4300033     char   *user_home;
4300034     char   *user_shell;
4300035     uid_t   uid;
4300036     uid_t   euid;
4300037     int     fd;
4300038     ssize_t size_read;
4300039     int     b;          // Index inside buffer.
4300040     int     loop;
4300041     char   *exec_argv[2];
4300042     int     status;
4300043     char   *tty_path;
4300044     //
4300045     // Check if login is running correctly.
4300046     //
4300047     euid = geteuid ();
4300048     uid  = geteuid ();
4300049     //  //
4300050     //  // Show process info.
4300051     //  //
4300052     //  heap_clear ();
4300053     //  process_info ();
4300054     //
```



```
4300055     // Check privileges.
4300056     //
4300057     if (!(uid == 0 && euid == 0))
4300058       {
4300059         printf ("%s: can only run with root privileges!\n", argv[0]);
4300060         exit (-1);
4300061       }
4300062     //
4300063     // Prepare arguments for the shell call.
4300064     //
4300065     exec_argv[0] = "-";
4300066     exec_argv[1] = NULL;
4300067     //
4300068     // Login.
4300069     //
4300070     while (1)
4300071       {
4300072         fd = open ("/etc/passwd", O_RDONLY);
4300073         //
4300074         if (fd < 0)
4300075           {
4300076             perror ("Cannot open file '/etc/passwd'");
4300077             exit (-1);
4300078           }
4300079         //
4300080         printf ("Log in as \"root\" or \"user\" "
4300081                 "with password \"ciao\" :-)\n");
4300082         input_line (login, "login:", LOGIN_MAX, INPUT_LINE_ECHO);
4300083         //
4300084         //
4300085         //
4300086         loop = 1;
4300087         while (loop)
4300088           {
4300089             for (b = 0; b < LINE_MAX; b++)
4300090               {
4300091                 size_read = read (fd, &buffer[b], (size_t) 1);
4300092                 if (size_read <= 0)
4300093                   {
4300094                     buffer[b] = 0;
4300095                     loop = 0;           // Close the middle loop.
4300096                     break;
4300097                   }
4300098                 if (buffer[b] == '\n')
4300099                   {
4300100                     buffer[b] = 0;
4300101                     break;
4300102                   }
4300103               }
4300104             //
4300105             user_name        = strtok (buffer, ":");
4300106             user_password    = strtok (NULL, ":");
4300107             user_uid         = strtok (NULL, ":");
4300108             user_gid         = strtok (NULL, ":");
4300109             user_description = strtok (NULL, ":");
4300110             user_home        = strtok (NULL, ":");
4300111             user_shell       = strtok (NULL, ":");
4300112             //
4300113             if (strcmp (user_name, login) == 0)
4300114               {
4300115                 input_line (password, "password:", PASSWORD_MAX,
4300116                             INPUT_LINE_STARS);
4300117                 //
4300118                 // Compare passwords: empty passwords are not allowed.
4300119                 //
4300120                 if (strcmp (user_password, password) == 0)
4300121                   {
4300122                     uid  = atoi (user_uid);
4300123                     euid = uid;
4300124                     //
4300125                     // Find the controlling terminal and change
4300126                     // property and access permissions.
4300127                     //
4300128                     tty_path = ttyname (STDIN_FILENO);
4300129                     if (tty_path != NULL)
4300130                       {
4300131                         status = chown (tty_path, uid, 0);
4300132                         if (status != 0)
4300133                           {
4300134                             perror (NULL);
4300135                           }
4300136                         status = chmod (tty_path, 0600);
4300137                         if (status != 0)
4300138                           {
4300139                             perror (NULL);
4300140                           }
4300141                       }
4300142                     //
4300143                     // Cd to the home directory, if present.
4300144                     //
4300145                     status = chdir (user_home);
4300146                     if (status != 0)
4300147                       {
4300148                         perror (NULL);
4300149                       }
4300150                     //
4300151                     // Now change personality.
4300152                     //
4300153                     setuid (uid);
4300154                     seteuid (euid);
4300155                     //
```

```
                    // Run the shell, replacing the login process; the
                    // environment is taken from 'init'.
                    //
                    execve (user_shell, exec_argv, envp);
                    exit (0);
                  }
                //
                // Login failed: will try again.
                //
                loop = 0;               // Close the middle loop.
                break;
              }
          }
        close (fd);
      }
}
```

applic/ls.c

«

Si veda la sezione u0.13.

```
#include <sys/os16.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
#include <dirent.h>
#include <pwd.h>
#include <time.h>

#define BUFFER_SIZE     16384
#define LIST_SIZE       256

//------------------------------------------------------------------------
static void usage         (void);
//------------------------------------------------------------------------
//------------------------------------------------------------------------
int compare (void *p1, void *p2);
//------------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    int            option_a = 0;
    int            option_l = 0;
    int            opt;
//  extern char   *optarg;              // not used.
    extern int     optind;
    extern int     optopt;
    struct stat    file_status;
    DIR           *dp;
    struct dirent *dir;
    char           buffer[BUFFER_SIZE];
    int            b;                   // Buffer index.
    char          *list[LIST_SIZE];
    int            l;                   // List index.
    int            len;                 // Name length.
    char          *path = NULL;
    char           pathname[PATH_MAX];
    struct passwd *pws;
    struct tm     *tms;
    //
    // Check for options.
    //
    while ((opt = getopt (argc, argv, ":al")) != -1)
      {
        switch (opt)
          {
            case 'l':
                option_l = 1;
                break;
            case 'a':
                option_a = 1;
                break;
            case '?':
                fprintf (stderr, "Unknown option -%c.\n", optopt);
                usage ();
                return (1);
                break;
            case ':':
                fprintf (stderr, "Missing argument for option -%c\n",
                         optopt);
                usage ();
                return (1);
                break;
            default:
                fprintf (stderr, "Getopt problem: unknown option %c\n",
                         opt);
                return (1);
          }
      }
    //
    // If no arguments are present, at least the current directory is
    // read.
    //
```



```
    if (optind == argc)
      {
        //
        // There are no more arguments. Replace the program name,
        // corresponding to 'argv[0]', with the current directory
        // path string.
        //
        argv[0] = ".";
        argc    = 1;
        optind  = 0;
      }
    //
    // This is a very simplified 'ls': if there is only a name
    // and it is a directory, the directory content is taken as
    // the new 'argv[]' array.
    //
    if (optind == (argc -1))
      {
        //
        // There is a request for a single name. Test if it exists
        // and if it is a directory.
        //
        if (stat(argv[optind], &file_status) != 0)
          {
            fprintf (stderr, "File \"%s\" does not exist!\n",
                     argv[optind]);
            return (2);
          }
        //
        if (S_ISDIR (file_status.st_mode))
          {
            //
            // Save the directory inside the 'path' pointer.
            //
            path = argv[optind];
            //
            // Open the directory.
            //
            dp = opendir (argv[optind]);
            if (dp == NULL)
              {
                perror (argv[optind]);
                return (3);
              }
            //
            // Read the directory and fill the buffer with names.
            //
            b = 0;
            l = 0;
            while ((dir = readdir (dp)) != NULL)
              {
                len = strlen (dir->d_name);
                //
                // Check if the buffer can hold it.
                //
                if ((b + len + 1 ) > BUFFER_SIZE)
                  {
                    fprintf (stderr, "not enough memory\n");
                    break;
                  }
                //
                // Consider the directory item only if there is
                // a valid name. If it is empty, just ignore it.
                //
                if (len > 0)
                  {
                    strcpy (&buffer[b], dir->d_name);
                    list[l] = &buffer[b];
                    b += len + 1;
                    l++;
                  }
              }
            //
            // Close the directory.
            //
            closedir (dp);
            //
            // Sort the list.
            //
            qsort (list, (size_t) l, sizeof (char *), compare);

            //
            // Convert the directory list into a new 'argv[]' array,
            // with a valid 'argc'. The variable 'optind' must be
            // reset to the first element index, because there is
            // no program name inside the new 'argv[]' at index zero.
            //
            argv   = list;
            argc   = l;
            optind = 0;
          }
      }
    //
    // Scan arguments, or list converted into 'argv[]'.
    //
    for (; optind < argc; optind++)
      {
        if (argv[optind][0] == '.')
          {
            //
            // Current name starts with '.'.
```

```
4310182             //
4310183             if (!option_a)
4310184               {
4310185                 //
4310186                 // Do not show name starting with '.'.
4310187                 //
4310188                 continue;
4310189               }
4310190           }
4310191         //
4310192         // Build the pathname.
4310193         //
4310194         if (path == NULL)
4310195           {
4310196             strcpy (&pathname[0], argv[optind]);
4310197           }
4310198         else
4310199           {
4310200             strcpy (pathname, path);
4310201             strcat (pathname, "/");
4310202             strcat (pathname, argv[optind]);
4310203           }
4310204         //
4310205         // Check if file exists, reading status.
4310206         //
4310207         if (stat(pathname, &file_status) != 0)
4310208           {
4310209             fprintf (stderr, "File \"%s\" does not exist!\n",
4310210                      pathname);
4310211             return (2);
4310212           }
4310213         //
4310214         // Show file name.
4310215         //
4310216         if (option_l)
4310217           {
4310218             //
4310219             // Print the file type.
4310220             //
4310221             if      (S_ISBLK  (file_status.st_mode)) printf ("b");
4310222             else if (S_ISCHR  (file_status.st_mode)) printf ("c");
4310223             else if (S_ISFIFO (file_status.st_mode)) printf ("p");
4310224             else if (S_ISREG  (file_status.st_mode)) printf ("-");
4310225             else if (S_ISDIR  (file_status.st_mode)) printf ("d");
4310226             else if (S_ISLNK  (file_status.st_mode)) printf ("l");
4310227             else if (S_ISSOCK (file_status.st_mode)) printf ("s");
4310228             else                                     printf ("?");
4310229             //
4310230             // Print permissions.
4310231             //
4310232             if (S_IRUSR & file_status.st_mode) printf ("r");
4310233             else printf ("-");
4310234             if (S_IWUSR & file_status.st_mode) printf ("w");
4310235             else printf ("-");
4310236             if (S_IXUSR & file_status.st_mode) printf ("x");
4310237             else printf ("-");
4310238             if (S_IRGRP & file_status.st_mode) printf ("r");
4310239             else printf ("-");
4310240             if (S_IWGRP & file_status.st_mode) printf ("w");
4310241             else printf ("-");
4310242             if (S_IXGRP & file_status.st_mode) printf ("x");
4310243             else printf ("-");
4310244             if (S_IROTH & file_status.st_mode) printf ("r");
4310245             else printf ("-");
4310246             if (S_IWOTH & file_status.st_mode) printf ("w");
4310247             else printf ("-");
4310248             if (S_IXOTH & file_status.st_mode) printf ("x");
4310249             else printf ("-");
4310250             //
4310251             // Print links.
4310252             //
4310253             printf (" %3i", (int) file_status.st_nlink);
4310254             //
4310255             // Print owner.
4310256             //
4310257             pws = getpwuid (file_status.st_uid);
4310258             //
4310259             printf (" %s", pws->pw_name);
4310260             //
4310261             // Print group (no group available);
4310262             //
4310263             printf (" (no group)");
4310264             //
4310265             // Print file size or device major-minor.
4310266             //
4310267             if (S_ISBLK (file_status.st_mode)
4310268                 || S_ISCHR (file_status.st_mode))
4310269               {
4310270                 printf (" %3i,", (int) major (file_status.st_rdev));
4310271                 printf (" %3i",  (int) minor (file_status.st_rdev));
4310272               }
4310273             else
4310274               {
4310275                 printf (" %8i", (int) file_status.st_size);
4310276               }
4310277             //
4310278             // Print modification date and time.
4310279             //
4310280             tms = localtime (&(file_status.st_mtime));
4310281             printf (" %4u-%02u-%02u %02u:%02u",
4310282                     tms->tm_year, tms->tm_mon, tms->tm_mday,
```



```
4310283                     tms->tm_hour, tms->tm_min);
4310284             //
4310285             // Print file name, but with no additional path.
4310286             //
4310287             printf (" %s\n", argv[optind]);
4310288           }
4310289         else
4310290           {
4310291             //
4310292             // Just show the file name and go to the next line.
4310293             //
4310294             printf ("%s\n", argv[optind]);
4310295           }
4310296       }
4310297     //
4310298     // All done.
4310299     //
4310300     return (0);
4310301 }
4310302 //---------------------------------------------------------------------
4310303 static void
4310304 usage (void)
4310305 {
4310306     fprintf (stderr, "Usage: ls [OPTION] [FILE]...\n");
4310307 }
4310308 //---------------------------------------------------------------------
4310309 int
4310310 compare (void *p1, void *p2)
4310311 {
4310312     char **pp1 = p1;
4310313     char **pp2 = p2;
4310314     //
4310315     return (strcmp (*pp1, *pp2));
4310316 }
4310317
```

applic/man.c

Si veda la sezione u0.14.

```
4320001 #include <unistd.h>
4320002 #include <stdlib.h>
4320003 #include <errno.h>
4320004 //---------------------------------------------------------------------
4320005 #define MAX_LINES   20
4320006 #define MAX_COLUMNS 80
4320007 //---------------------------------------------------------------------
4320008 static char *man_page_directory = "/usr/share/man";
4320009 //---------------------------------------------------------------------
4320010 static void  usage          (void);
4320011 static FILE *open_man_page  (int section, char *name);
4320012 static void build_path_name (int section, char *name, char *path);
4320013 //---------------------------------------------------------------------
4320014 int
4320015 main (int argc, char *argv[], char *envp[])
4320016 {
4320017     FILE       *fp;
4320018     char       *name;
4320019     int         section;
4320020     int         c;
4320021     int         line   = 1; // Line internal counter.
4320022     int         column = 1; // Column internal counter.
4320023     int         loop;
4320024     //
4320025     // There must be minimum an argument, and maximum two.
4320026     //
4320027     if (argc < 2 || argc > 3)
4320028       {
4320029         usage ();
4320030         return (1);
4320031       }
4320032     //
4320033     // If there are two arguments, there must be the
4320034     // section number.
4320035     //
4320036     if (argc == 3)
4320037       {
4320038         section = atoi (argv[1]);
4320039         name = argv[2];
4320040       }
4320041     else
4320042       {
4320043         section = 0;
4320044         name = argv[1];
4320045       }
4320046     //
4320047     // Try to open the manual page.
4320048     //
4320049     fp = open_man_page (section, name);
4320050     //
4320051     if (fp == NULL)
4320052       {
4320053         //
4320054         // Error opening file.
4320055         //
4320056         return (1);
4320057       }
4320058
4320059     //
4320060     // The following loop continues while the file
4320061     // gives characters, or when a command to change
```

```
    // file or to quit is given.
    //
    for (loop = 1; loop; )
      {
        //
        // Read a single character.
        //
        c = getc (fp);
        //
        if (c == EOF)
          {
            loop = 0;
            break;
          }
        //
        // If the character read is a special one,
        // the line/column calculation is modified,
        // so that it is known when to stop scrolling.
        //
        switch (c)
          {
            case '\r':
                //
                // Displaying this character, the cursor should go
                // back to the first column. So the column counter
                // is reset.
                //
                column = 1;
                break;
            case '\n':
                //
                // Displaying this character, the cursor should go
                // back to the next line, at the first column.
                // So the column counter is reset and the line
                // counter is incremented.
                //
                line++;
                column = 1;
                break;
            case '\b':
                //
                // Displaying this character, the cursor should go
                // back one position, unless it is already at the
                // beginning.
                //
                if (column > 1)
                  {
                    column--;
                  }
                break;
            default:
                //
                // Any other character must increase the column
                // counter.
                //
                column++;
          }
        //
        // Display the character, even if it is a special one:
        // it is responsibility of the screen device management
        // to do something good with special characters.
        //
        putchar (c);
        //
        // If the column counter is gone beyond the screen columns,
        // then adjust the column counter and increment the line
        // counter.
        //
        if (column > MAX_COLUMNS)
          {
            column -= MAX_COLUMNS;
            line++;
          }
        //
        // Check if there is space for scrolling.
        //
        if (line < MAX_LINES)
          {
            continue;
          }
        //
        // Here, displayed lines are MAX_LINES.
        //
        if (column > 1)
          {
            //
            // Something was printed at the current line: must
            // do a new line.
            //
            putchar ('\n');
          }
        //
        // Show the more prompt.
        //
        printf ("--More--");
        fflush (stdout);
        //
        // Read a character from standard input.
        //
        c = getchar ();
        //
```



```
        // Consider command 'q', but any other character
        // can be introduced, to let show the next page.
        //
        switch (c)
          {
            case 'Q':
            case 'q':
                //
                // Quit. But must erase the '--More--' prompt.
                //
                printf ("\b \b\b \b\b \b\b \b\b \b");
                printf ("\b \b\b \b\b \b\b \b");
                fclose (fp);
                return (0);
          }
        //
        // Backspace to overwrite '--More--' and the character
        // pressed.
        //
        printf ("\b \b\b \b\b \b\b \b\b \b\b \b\b \b\b \b");
        //
        // Reset line/column counters.
        //
        column = 1;
        line = 1;
      }
    //
    // Close the file pointer if it is still open.
    //
    if (fp != NULL)
      {
        fclose (fp);
      }
    //
    return (0);
}
//-----------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage: man [SECTION] NAME\n");
}
//-----------------------------------------------------------------------
FILE *
open_man_page (int section, char *name)
{
    FILE       *fp;
    char        path[PATH_MAX];
    struct stat file_status;
    //
    //
    //
    if (section > 0)
      {
        build_path_name (section, name, path);
        //
        // Check if file exists.
        //
        if (stat (path, &file_status) != 0)
          {
            fprintf (stderr, "Man page %s(%i) does not exist!\n",
                     name, section);
            return (NULL);
          }
      }
    else
      {
        //
        // Must try a section.
        //
        for (section = 1; section < 9; section++)
          {
            build_path_name (section, name, path);
            //
            // Check if file exists.
            //
            if (stat (path, &file_status) == 0)
              {
                //
                // Found.
                //
                break;
              }
          }
      }
    //
    // Check if a file was found.
    //
    if (section < 9)
      {
        fp = fopen (path, "r");
        //
        if (fp == NULL)
          {
            //
            // Error opening file.
            //
            perror (path);
            return (NULL);
          }
        else
```

```
          {
            //
            // Opened right.
            //
            return (fp);
          }
      }
    else
      {
        fprintf (stderr, "Man page %s does not exist!\n",
                 name);
        return (NULL);
      }
}
//---------------------------------------------------------------------
void
build_path_name (int section, char *name, char *path)
{
    char  string_section[10];
    //
    // Convert the section number into a string.
    //
    sprintf (string_section, "%i", section);
    //
    // Prepare the path to the man file.
    //
    path[0] = 0;
    strcat (path, man_page_directory);
    strcat (path, "/");
    strcat (path, name);
    strcat (path, ".");
    strcat (path, string_section);
}
```

applic/mkdir.c

«

Si veda la sezione u0.15.

```
#include <sys/os16.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
//---------------------------------------------------------------------
static int  mkdir_parents (const char *path, mode_t mode);
static void usage         (void);
//---------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    sysmsg_uarea_t msg;
    int            status;
    mode_t         mode     = 0;
    int            m;           // Index inside mode argument.
    int            digit;
    char         **dir;
    int            d;           // Directory index.
    int            option_p = 0;
    int            option_m = 0;
    int            opt;
    extern char   *optarg;
    extern int     optind;
    extern int     optopt;
    //
    // There must be at least an argument, plus the program name.
    //
    if (argc < 2)
      {
        usage ();
        return (1);
      }
    //
    // Check for options, starting from 'p'. The 'dir' pointer is used
    // to calculate the argument pointer to the first directory [1].
    // The macro-instruction 'max()' is declared inside <sys/os16.h>
    // and does the expected thing.
    //
    while ((opt = getopt (argc, argv, ":pm:")) != -1)
      {
        switch (opt)
          {
            case 'm':
                option_m = 1;
                for (m = 0; m < strlen (optarg); m++)
                  {
                    digit = (optarg[m] - '0');
                    if (digit < 0 || digit > 7)
                      {
                        usage ();
                        return (2);
                      }
                    mode = mode * 8 + digit;
                  }
                break;
```



```
            case 'p':
                option_p = 1;
                break;
            case '?':
                printf ("Unknown option -%c.\n", optopt);
                usage ();
                return (1);
                break;
            case ':':
                printf ("Missing argument for option -%c\n", optopt);
                usage ();
                return (2);
                break;
            default:
                printf ("Getopt problem: unknown option %c\n", opt);
                return (3);
          }
      }
    //
    dir = argv + optind;
    //
    // Check if the mode is to be set to a default value.
    //
    if (!option_m)
      {
        //
        // Default mode.
        //
        sys (SYS_UAREA, &msg, (sizeof msg));
        mode = 0777 & ~msg.umask;
      }
    //
    // Directory creation.
    //
    for (d = 0; dir[d] != NULL; d++)
      {
        if (option_p)
          {
            status = mkdir_parents (dir[d], mode);
            if (status != 0)
             {
                perror (dir[d]);
                return (3);
             }
          }
        else
          {
            status = mkdir (dir[d], mode);
            if (status != 0)
              {
                perror (dir[d]);
                return (4);
              }
          }
      }
    //
    // All done.
    //
    return (0);
}
//---------------------------------------------------------------------
static int
mkdir_parents (const char *path, mode_t mode)
{
    char        path_copy[PATH_MAX];
    char       *path_parent;
    struct stat fst;
    int         status;
    //
    // Check if the path is empty.
    //
    if (path == NULL || strlen (path) == 0)
      {
        //
        // Recursion ends here.
        //
        return (0);
      }
    //
    // Check if it does already exists.
    //
    status = stat (path, &fst);
    if (status == 0 && fst.st_mode & S_IFDIR)
      {
        //
        // The path exists and is a directory.
        //
        return (0);
      }
    else if (status == 0 && !(fst.st_mode & S_IFDIR))
      {
        //
        // The path exists but is not a directory.
        //
        errno = ENOTDIR;        // Not a directory.
        return (-1);
      }
    //
    // Get the directory path.
    //
    strncpy (path_copy, path, PATH_MAX);
```

```
    path_parent = dirname (path_copy);
    //
    // If it is '.', or '/', the recursion is terminated.
    //
    if (strncmp (path_parent, ".", PATH_MAX) == 0  ||
        strncmp (path_parent, "/", PATH_MAX) == 0)
      {
        return (0);
      }
    //
    // Otherwise, continue the recursion.
    //
    status = mkdir_parents (path_parent, mode);
    if (status != 0)
      {
        return (-1);
      }
    //
    // Previous directories are there: create the current one.
    //
    status = mkdir (path, mode);
    if (status)
      {
        perror (path);
        return (-1);
      }

    return (0);
}
//-----------------------------------------------------------------------
static void
usage (void)
{
    fprintf (stderr, "Usage: mkdir [-p] [-m OCTAL_MODE] DIR...\n");
}
```

applic/more.c

« Si veda la sezione u0.16.

```
#include <unistd.h>
#include <errno.h>
//-----------------------------------------------------------------------
#define MAX_LINES   20
#define MAX_COLUMNS 80
//-----------------------------------------------------------------------
static void usage (void);
//-----------------------------------------------------------------------
int
main (int argc, char *argv[], char *envp[])
{
    FILE   *fp;
    char   *name;
    int     c;
    int     line   = 1; // Line internal counter.
    int     column = 1; // Column internal counter.
    int     a;          // Index inside arguments.
    int     loop;
    //
    // There must be at least an argument, plus the program name.
    //
    if (argc < 2)
      {
        usage ();
        return (1);
      }
    //
    // No options are allowed.
    //
    for (a = 1; a < argc ; a++)
      {
        //
        // Get next name from arguments.
        //
        name = argv[a];
        //
        // Try to open the file, read only.
        //
        fp = fopen (name, "r");
        //
        if (fp == NULL)
          {
            //
            // Error opening file.
            //
            perror (name);
            return (1);
          }
        //
        // Print the file name to be displayed.
        //
        printf ("== %s ==\n", name);
        line++;
        //
        // The following loop continues while the file
        // gives characters, or when a command to change
        // file or to quit is given.
        //
        for (loop = 1; loop; )
          {
            //
```



```
            // Read a single character.
            //
            c = getc (fp);
            //
            if (c == EOF)
              {
                loop = 0;
                break;
              }
            //
            // If the character read is a special one,
            // the line/column calculation is modified,
            // so that it is known when to stop scrolling.
            //
            switch (c)
              {
                case '\r':
                    //
                    // Displaying this character, the cursor should go
                    // back to the first column. So the column counter
                    // is reset.
                    //
                    column = 1;
                    break;
                case '\n':
                    //
                    // Displaying this character, the cursor should go
                    // back to the next line, at the first column.
                    // So the column counter is reset and the line
                    // counter is incremented.
                    //
                    line++;
                    column = 1;
                    break;
                case '\b':
                    //
                    // Displaying this character, the cursor should go
                    // back one position, unless it is already at the
                    // beginning.
                    //
                    if (column > 1)
                      {
                        column--;
                      }
                    break;
                default:
                    //
                    // Any other character must increase the column
                    // counter.
                    //
                    column++;
              }
            //
            // Display the character, even if it is a special one:
            // it is responsibility of the screen device management
            // to do something good with special characters.
            //
            putchar (c);
            //
            // If the column counter is gone beyond the screen columns,
            // then adjust the column counter and increment the line
            // counter.
            //
            if (column > MAX_COLUMNS)
              {
                column -= MAX_COLUMNS;
                line++;
              }
            //
            // Check if there is space for scrolling.
            //
            if (line < MAX_LINES)
              {
                continue;
              }
            //
            // Here, displayed lines are MAX_LINES.
            //
            if (column > 1)
              {
                //
                // Something was printed at the current line: must
                // do a new line.
                //
                putchar ('\n');
              }
            //
            // Show the more prompt.
            //
            printf ("--More--");
            fflush (stdout);
            //
            // Read a character from standard input.
            //
            c = getchar ();
            //
            // Consider commands 'n' and 'q', but any other character
            // can be introduced, to let show the next page.
            //
            switch (c)
              {
```

```
4340163                 case 'N':
4340164                 case 'n':
4340165                     //
4340166                     // Go to the next file, if any.
4340167                     //
4340168                     fclose (fp);
4340169                     fp = NULL;
4340170                     loop = 0;
4340171                     break;
4340172                 case 'Q':
4340173                 case 'q':
4340174                     //
4340175                     // Quit. But must erase the '--More--' prompt.
4340176                     //
4340177                     printf ("\b \b\b \b\b \b\b \b\b \b");
4340178                     printf ("\b \b\b \b\b \b\b \b");
4340179                     fclose (fp);
4340180                     return (0);
4340181               }
4340182             //
4340183             // Backspace to overwrite '--More--' and the character
4340184             // pressed.
4340185             //
4340186             printf ("\b \b\b \b\b \b\b \b\b \b\b \b\b \b\b \b\b \b");
4340187             //
4340188             // Reset line/column counters.
4340189             //
4340190             column = 1;
4340191             line = 1;
4340192           }
4340193         //
4340194         // Close the file pointer if it is still open.
4340195         //
4340196         if (fp != NULL)
4340197           {
4340198             fclose (fp);
4340199           }
4340200       }
4340201     //
4340202     return (0);
4340203 }
4340204 //---------------------------------------------------------------------
4340205 static void
4340206 usage (void)
4340207 {
4340208     fprintf (stderr, "Usage: more FILE...\n");
4340209 }
```

applic/mount.c

«

Si veda la sezione u0.4.

```
4350001 #include <unistd.h>
4350002 #include <stdlib.h>
4350003 #include <sys/stat.h>
4350004 #include <sys/types.h>
4350005 #include <fcntl.h>
4350006 #include <errno.h>
4350007 #include <signal.h>
4350008 #include <stdio.h>
4350009 #include <sys/wait.h>
4350010 #include <stdio.h>
4350011 #include <string.h>
4350012 #include <limits.h>
4350013 #include <sys/os16.h>
4350014 //---------------------------------------------------------------------
4350015 static void usage (void);
4350016 //---------------------------------------------------------------------
4350017 int
4350018 main (int argc, char *argv[], char *envp[])
4350019 {
4350020     int     options;
4350021     int     status;
4350022     //
4350023     //
4350024     //
4350025     if (argc < 3 || argc > 4)
4350026       {
4350027         usage ();
4350028         return (1);
4350029       }
4350030     //
4350031     // Set options.
4350032     //
4350033     if (argc == 4)
4350034       {
4350035         if      (strcmp (argv[3], "rw") == 0)
4350036           {
4350037             options = MOUNT_DEFAULT;
4350038           }
4350039         else if (strcmp (argv[3], "ro") == 0)
4350040           {
4350041             options = MOUNT_RO;
4350042           }
4350043         else
4350044           {
4350045             printf ("Invalid mount option: only \"ro\" or \"rw\" "
4350046                     "are allowed\n");
4350047             return (2);
4350048           }
4350049       }
```



```
4350050     else
4350051       {
4350052         options = MOUNT_DEFAULT;
4350053       }
4350054     //
4350055     // System call.
4350056     //
4350057     status = mount (argv[1], argv[2], options);
4350058     if (status != 0)
4350059       {
4350060         perror (NULL);
4350061         return (2);
4350062       }
4350063     //
4350064     return (0);
4350065 }
4350066 //---------------------------------------------------------------------
4350067 static void
4350068 usage (void)
4350069 {
4350070     fprintf (stderr, "Usage: mount DEVICE MOUNT_POINT "
4350071                      "[MOUNT_OPTIONS]\n");
4350072 }
```

applic/ps.c

«

Si veda la sezione u0.17.

```
4360001 #include <kernel/proc.h>
4360002 #include <unistd.h>
4360003 #include <stdio.h>
4360004 #include <fcntl.h>
4360005 #include <unistd.h>
4360006 #include <stdlib.h>
4360007 //---------------------------------------------------------------------
4360008 void
4360009 print_proc_head (void)
4360010 {
4360011     printf (
4360012 "pp  p pg                                                        \n"
4360013 "id id rp  tty  uid euid suid usage s iaddr isiz daddr dsiz sp   name\n"
4360014           );
4360015 }
4360016 //---------------------------------------------------------------------
4360017 void
4360018 print_proc_pid (proc_t *ps, pid_t pid)
4360019 {
4360020     char  stat;
4360021     switch (ps->status)
4360022       {
4360023         case PROC_EMPTY   : stat = '-'; break;
4360024         case PROC_CREATED : stat = 'c'; break;
4360025         case PROC_READY   : stat = 'r'; break;
4360026         case PROC_RUNNING : stat = 'R'; break;
4360027         case PROC_SLEEPING: stat = 's'; break;
4360028         case PROC_ZOMBIE  : stat = 'z'; break;
4360029         default           : stat = '?'; break;
4360030       }
4360031
4360032     printf ("%2i %2i %2i %04x %4i %4i %4i %02i.%02i %c %05lx %04x ",
4360033               (unsigned int) ps->ppid,
4360034               (unsigned int) pid,
4360035               (unsigned int) ps->pgrp,
4360036               (unsigned int) ps->device_tty,
4360037               (unsigned int) ps->uid,
4360038               (unsigned int) ps->euid,
4360039               (unsigned int) ps->suid,
4360040               (unsigned int) ((ps->usage / CLOCKS_PER_SEC) / 60),
4360041               (unsigned int) ((ps->usage / CLOCKS_PER_SEC) % 60),
4360042               stat,
4360043               (unsigned long int) ps->address_i,
4360044               (unsigned int) ps->size_i);
4360045
4360046     printf ("%05lx %04x %04x %s",
4360047               (unsigned long int) ps->address_d,
4360048               (unsigned int) ps->size_d,
4360049               (unsigned int) ps->sp,
4360050                              ps->name);
4360051
4360052     printf ("\n");
4360053 }
4360054 //---------------------------------------------------------------------
4360055 int
4360056 main (void)
4360057 {
4360058     pid_t   pid;
4360059     proc_t *ps;
4360060     int     fd;
4360061     ssize_t size_read;
4360062     char    buffer[sizeof (proc_t)];
4360063
4360064     fd = open ("/dev/kmem_ps", O_RDONLY);
4360065     if (fd < 0)
4360066       {
4360067         perror ("ps: cannot open \"/dev/kmem_ps\"");
4360068         exit (0);
4360069       }
4360070
4360071     print_proc_head ();
4360072     for (pid = 0; pid < PROCESS_MAX; pid++)
4360073       {
```

```
4360074         lseek (fd, (off_t) pid, SEEK_SET);
4360075         size_read = read (fd, buffer, sizeof (proc_t));
4360076         if (size_read < sizeof (proc_t))
4360077           {
4360078             printf ("ps: cannot read \"/dev/kmem_ps\" pid %i", pid);
4360079             perror (NULL);
4360080             continue;
4360081           }
4360082         ps = (proc_t *) buffer;
4360083         if (ps->status > 0)
4360084           {
4360085             ps->name[PATH_MAX-1] = 0; // Terminated string.
4360086             print_proc_pid (ps, pid);
4360087           }
4360088       }
4360089
4360090     close (fd);
4360091     return (0);
4360092 }
```

applic/rm.c

«

Si veda la sezione u0.18.

```
4370001 #include <fcntl.h>
4370002 #include <sys/stat.h>
4370003 #include <stddef.h>
4370004 #include <unistd.h>
4370005 #include <errno.h>
4370006 //-------------------------------------------------------------------------
4370007 static void usage (void);
4370008 //-------------------------------------------------------------------------
4370009 int
4370010 main (int argc, char *argv[], char *envp[])
4370011 {
4370012     int         a;              // Argument index.
4370013     int         status;
4370014     struct stat file_status;
4370015     //
4370016     // No options are known, but at least an argument must be given.
4370017     //
4370018     if (argc < 2)
4370019       {
4370020         usage ();
4370021         return (1);
4370022       }
4370023     //
4370024     // Scan arguments.
4370025     //
4370026     for(a = 1; a < argc; a++)
4370027       {
4370028         //
4370029         // Verify if the file exists.
4370030         //
4370031         if (stat(argv[a], &file_status) != 0)
4370032           {
4370033             fprintf (stderr, "File \"%s\" does not exist!\n",
4370034                      argv[a]);
4370035             continue;
4370036           }
4370037         //
4370038         // File exists: check the file type.
4370039         //
4370040         if (S_ISDIR (file_status.st_mode))
4370041           {
4370042             fprintf (stderr, "Cannot remove directory \"%s\"!\n",
4370043                      argv[a]);
4370044             continue;
4370045           }
4370046         //
4370047         // Can remove it.
4370048         //
4370049         status = unlink (argv[a]);
4370050         if (status != 0)
4370051           {
4370052             perror (NULL);
4370053             return (2);
4370054           }
4370055       }
4370056     return (0);
4370057 }
4370058 //-------------------------------------------------------------------------
4370059 static void
4370060 usage (void)
4370061 {
4370062     fprintf (stderr, "Usage: rm FILE...\n");
4370063 }
```

applic/shell.c

«

Si veda la sezione u0.19.

```
4380001 #include <unistd.h>
4380002 #include <stdlib.h>
4380003 #include <sys/stat.h>
4380004 #include <sys/types.h>
4380005 #include <fcntl.h>
4380006 #include <errno.h>
4380007 #include <unistd.h>
4380008 #include <signal.h>
```



```
4380009 #include <stdio.h>
4380010 #include <sys/wait.h>
4380011 #include <stdio.h>
4380012 #include <string.h>
4380013 #include <limits.h>
4380014 #include <sys/os16.h>
4380015 //-------------------------------------------------------------------------
4380016 #define PROMPT_SIZE     16
4380017 //-------------------------------------------------------------------------
4380018 static void sh_cd       (int argc, char *argv[]);
4380019 static void sh_pwd      (int argc, char *argv[]);
4380020 static void sh_umask    (int argc, char *argv[]);
4380021 //-------------------------------------------------------------------------
4380022 int
4380023 main (int argc, char *argv[], char *envp[])
4380024 {
4380025   char    buffer_cmd[ARG_MAX/2];
4380026   char   *argv_cmd[ARG_MAX/16];
4380027   char    prompt[PROMPT_SIZE];
4380028   uid_t   uid;
4380029   int     argc_cmd;
4380030   pid_t   pid_cmd;
4380031   pid_t   pid_dead;
4380032   int     status;
4380033   //
4380034   //
4380035   //
4380036   uid = geteuid ();
4380037   //
4380038   // Load processes, reading the keyboard.
4380039   //
4380040   while (1)
4380041     {
4380042       if (uid == 0)
4380043         {
4380044           strncpy (prompt, "# ", PROMPT_SIZE);
4380045         }
4380046       else
4380047         {
4380048           strncpy (prompt, "$ ", PROMPT_SIZE);
4380049         }
4380050       //
4380051       input_line (buffer_cmd, prompt, (ARG_MAX/2), INPUT_LINE_ECHO);
4380052       //
4380053       // Clear 'argv_cmd[]';
4380054       //
4380055       for (argc_cmd = 0; argc_cmd < (ARG_MAX/16); argc_cmd++)
4380056         {
4380057           argv_cmd[argc_cmd] = NULL;
4380058         }
4380059       //
4380060       // Initialize the command scan.
4380061       //
4380062       argv_cmd[0] = strtok (buffer_cmd, " \t");
4380063       //
4380064       // Verify: if the input is not valid, loop again.
4380065       //
4380066       if (argv_cmd[0] == NULL)
4380067         {
4380068           continue;
4380069         }
4380070       //
4380071       // Find the arguments.
4380072       //
4380073       for (argc_cmd = 1;
4380074            argc_cmd < ((ARG_MAX/16)-1) && argv_cmd[argc_cmd-1] != NULL;
4380075            argc_cmd++)
4380076         {
4380077           argv_cmd[argc_cmd] = strtok (NULL, " \t");
4380078         }
4380079       //
4380080       // If there are too many arguments, show a message and continue.
4380081       //
4380082       if (argv_cmd[argc_cmd-1] != NULL)
4380083         {
4380084           errset (E2BIG);             // Argument list too long.
4380085           perror (NULL);
4380086           continue;
4380087         }
4380088       //
4380089       // Correct the value for 'argc_cmd', because actually
4380090       // it counts also the NULL element.
4380091       //
4380092       argc_cmd--;
4380093       //
4380094       // Verify if it is an internal command.
4380095       //
4380096       if      (strcmp (argv_cmd[0], "exit") == 0)
4380097         {
4380098           return (0);
4380099         }
4380100       else if (strcmp (argv_cmd[0], "cd") == 0)
4380101         {
4380102           sh_cd (argc_cmd, argv_cmd);
4380103           continue;
4380104         }
4380105       else if (strcmp (argv_cmd[0], "pwd") == 0)
4380106         {
4380107           sh_pwd (argc_cmd, argv_cmd);
4380108           continue;
4380109         }
```

```
4380110       else if (strcmp (argv_cmd[0], "umask") == 0)
4380111         {
4380112           sh_umask (argc_cmd, argv_cmd);
4380113           continue;
4380114         }
4380115       //
4380116       // It should be a program to run.
4380117       //
4380118       pid_cmd = fork ();
4380119       if (pid_cmd == -1)
4380120         {
4380121           printf ("%s: cannot run command", argv[0]);
4380122           perror (NULL);
4380123         }
4380124       else if (pid_cmd == 0)
4380125         {
4380126           execvp (argv_cmd[0], argv_cmd);
4380127           perror (NULL);
4380128           exit (0);
4380129         }
4380130       while (1)
4380131         {
4380132           pid_dead = wait (&status);
4380133           if (pid_dead == pid_cmd)
4380134             {
4380135               break;
4380136             }
4380137         }
4380138       printf ("pid %i terminated with status %i.\n",
4380139               (int) pid_dead, status);
4380140     }
4380141 }
4380142 //---------------------------------------------------------------------
4380143 static void
4380144 sh_cd (int argc, char *argv[])
4380145 {
4380146   int status;
4380147   //
4380148   if (argc != 2)
4380149     {
4380150       errset (EINVAL);                // Invalid argument.
4380151       perror (NULL);
4380152       return;
4380153     }
4380154   //
4380155   status = chdir (argv[1]);
4380156   if (status != 0)
4380157     {
4380158       perror (NULL);
4380159     }
4380160   return;
4380161 }
4380162 //---------------------------------------------------------------------
4380163 static void
4380164 sh_pwd (int argc, char *argv[])
4380165 {
4380166   char  path[PATH_MAX];
4380167   void *pstatus;
4380168   //
4380169   if (argc != 1)
4380170     {
4380171       errset (EINVAL);                // Invalid argument.
4380172       perror (NULL);
4380173       return;
4380174     }
4380175   //
4380176   // Get the current directory.
4380177   //
4380178   pstatus = getcwd (path, (size_t) PATH_MAX);
4380179   if (pstatus == NULL)
4380180     {
4380181       perror (NULL);
4380182     }
4380183   else
4380184     {
4380185       printf ("%s\n", path);
4380186     }
4380187   return;
4380188 }
4380189 //---------------------------------------------------------------------
4380190 static void
4380191 sh_umask (int argc, char *argv[])
4380192 {
4380193   sysmsg_uarea_t msg;
4380194   char          *m;           // Index inside the umask octal string.
4380195   int            mask;
4380196   int            digit;
4380197   //
4380198   if (argc > 2)
4380199     {
4380200       errset (EINVAL);                // Invalid argument.
4380201       perror (NULL);
4380202       return;
4380203     }
4380204   //
4380205   // If no argument is available, the umask is shown, with a direct
4380206   // system call.
4380207   //
4380208   if (argc == 1)
4380209     {
4380210       sys (SYS_UAREA, &msg, (sizeof msg));
```



```
4380211       printf ("%04o\n", msg.umask);
4380212       return;
4380213     }
4380214   //
4380215   // Get the mask: must be the first argument.
4380216   //
4380217   for (mask = 0, m = argv[1]; *m != 0; m++)
4380218     {
4380219       digit = (*m - '0');
4380220       if (digit < 0 || digit > 7)
4380221         {
4380222           errset (EINVAL);            // Invalid argument.
4380223           perror (NULL);
4380224           return;
4380225         }
4380226       mask = mask * 8 + digit;
4380227     }
4380228   //
4380229   // Set the umask and return.
4380230   //
4380231   umask (mask);
4380232   return;
4380233 }
```

applic/touch.c

«

Si veda la sezione u0.20.

```
4390001 #include <fcntl.h>
4390002 #include <sys/stat.h>
4390003 #include <utime.h>
4390004 #include <stddef.h>
4390005 #include <unistd.h>
4390006 #include <errno.h>
4390007 //---------------------------------------------------------------------
4390008 static void usage (void);
4390009 //---------------------------------------------------------------------
4390010 int
4390011 main (int argc, char *argv[], char *envp[])
4390012 {
4390013     int         a;              // Argument index.
4390014     int         status;
4390015     struct stat file_status;
4390016     //
4390017     // No options are known, but at least an argument must be given.
4390018     //
4390019     if (argc < 2)
4390020       {
4390021         usage ();
4390022         return (1);
4390023       }
4390024     //
4390025     // Scan arguments.
4390026     //
4390027     for(a = 1; a < argc; a++)
4390028       {
4390029         //
4390030         // Verify if the file exists, through the return value of
4390031         // 'stat()'. No other checks are made.
4390032         //
4390033         if (stat(argv[a], &file_status) == 0)
4390034          {
4390035             //
4390036             // File exists: should be updated the times.
4390037             //
4390038             status = utime (argv[a], NULL);
4390039             if (status != 0)
4390040               {
4390041                 perror (NULL);
4390042                 return (2);
4390043               }
4390044          }
4390045         else
4390046           {
4390047             //
4390048             // File does not exist: should be created.
4390049             //
4390050             status = open (argv[a], O_WRONLY|O_CREAT|O_TRUNC, 0666);
4390051             //
4390052             if (status >= 0)
4390053               {
4390054                 //
4390055                 // Here, the variable 'status' is the file
4390056                 // descriptor to be closed.
4390057                 //
4390058                 status = close (status);
4390059                 if (status != 0)
4390060                   {
4390061                     perror (NULL);
4390062                     return (3);
4390063                   }
4390064               }
4390065             else
4390066               {
4390067                 perror (NULL);
4390068                 return (4);
4390069               }
4390070           }
4390071       }
4390072     return (0);
4390073 }
```

```
4390074 //---------------------------------------------------------------------
4390075 static void
4390076 usage (void)
4390077 {
4390078     fprintf (stderr, "Usage: touch FILE...\n");
4390079 }
```

applic/tty.c

«

Si veda la sezione u0.21.

```
4400001 #include <fcntl.h>
4400002 #include <sys/stat.h>
4400003 #include <utime.h>
4400004 #include <stddef.h>
4400005 #include <unistd.h>
4400006 #include <errno.h>
4400007 #include <sys/os16.h>
4400008 #include <sys/types.h>
4400009 //---------------------------------------------------------------------
4400010 static void     usage (void);
4400011 //---------------------------------------------------------------------
4400012 int
4400013 main (int argc, char *argv[], char *envp[])
4400014 {
4400015   int             dev_minor;
4400016   struct stat     file_status;
4400017   //
4400018   // No options and no arguments.
4400019   //
4400020   if (argc > 1)
4400021     {
4400022       usage ();
4400023       return (1);
4400024     }
4400025   //
4400026   // Verify the standard input.
4400027   //
4400028   if (fstat (STDIN_FILENO, &file_status) == 0)
4400029     {
4400030       if (major (file_status.st_rdev) == DEV_CONSOLE_MAJOR)
4400031         {
4400032           dev_minor = minor (file_status.st_rdev);
4400033           //
4400034           // If minor is equal to 0xFF, it is '/dev/console'
4400035           // that is not a controlling terminal, but just
4400036           // a reference for the current virtual console.
4400037           //
4400038           if (dev_minor < 0xFF)
4400039             {
4400040               printf ("/dev/console%i\n", dev_minor);
4400041             }
4400042         }
4400043     }
4400044   else
4400045     {
4400046       perror ("Cannot get standard input file status");
4400047       return (2);
4400048     }
4400049   //
4400050   return (0);
4400051 }
4400052
4400053 //---------------------------------------------------------------------
4400054 static void
4400055 usage (void)
4400056 {
4400057   fprintf (stderr, "Usage: tty\n");
4400058 }
```

applic/umount.c

«

Si veda la sezione u0.4.

```
4410001 #include <unistd.h>
4410002 #include <stdlib.h>
4410003 #include <sys/stat.h>
4410004 #include <sys/types.h>
4410005 #include <fcntl.h>
4410006 #include <errno.h>
4410007 #include <signal.h>
4410008 #include <stdio.h>
4410009 #include <sys/wait.h>
4410010 #include <stdio.h>
4410011 #include <string.h>
4410012 #include <limits.h>
4410013 #include <sys/os16.h>
4410014 //---------------------------------------------------------------------
4410015 static void usage (void);
4410016 //---------------------------------------------------------------------
4410017 int
4410018 main (int argc, char *argv[], char *envp[])
4410019 {
4410020     int     status;
4410021     //
4410022     // One argument is mandatory.
4410023     //
4410024     if (argc != 2)
4410025       {
4410026         usage ();
4410027         return (1);
4410028       }
4410029     //
4410030     // System call.
4410031     //
4410032     status = umount (argv[1]);
4410033     if (status != 0)
4410034       {
4410035         perror (argv[1]);
4410036         return (2);
4410037       }
4410038     //
4410039     return (0);
4410040 }
4410041 //---------------------------------------------------------------------
4410042 static void
4410043 usage (void)
4410044 {
4410045     fprintf (stderr, "Usage: umount MOUNT_POINT\n");
4410046 }
```

Parte xxxvii

Un sistema operativo giocattolo, denominato «05»

Preparazione 1903
File-immagine 1903
Directory di lavoro 1905
Directory «05/» 1905
Script di collegamento 1907
Altre directory 1908
Libreria standard per iniziare 1909
Libreria «limits.h» 1909
File isolati per dichiarazioni riprese in più librerie 1911
Libreria «stdbool.h» 1911
Libreria «time.h» 1912
Libreria «ctype.h» 1912
Libreria «stdint.h» 1913
Libreria «inttypes.h» 1914
Libreria «stdarg.h» 1917
Libreria «stddef.h» 1917
Libreria «stdlib.h» 1917
Libreria «string.h» 1918
Libreria «stdio.h» 1920
Librerie specifiche generali 1931
File «build.h» 1931
Libreria «io.h» 1931
Libreria «multiboot.h» 1932
File «os.h» 1933
Libreria «vga.h» 1935
Un primo kernel di prova 1939
File «kernel.h» 1939
Altri file mancanti 1942
Compilazione e prova di funzionamento 1942
Tabella GDT 1943
Struttura 1943
Libreria «gdt.h» 1943
Modifiche da apportare a «kernel_main.c» 1946
Gestione della memoria 1947
Gestione della memoria attraverso una lista 1947
Libreria «mm.h» 1948
Funzioni per l'allocazione della memoria 1950
Verifica del funzionamento 1953
Tabella IDT 1955
File di intestazione «int.h» e file delle routine di interruzione «isr.s» 1955
Funzioni per definire la tabella IDT 1960
Gestione delle interruzioni 1963
Piccole funzioni di contorno 1965
Verifica del funzionamento 1965
Chiamate di sistema 1967
File di intestazione «syscall.h» 1967
Fasi successive all'interruzione 1967
Verifica del funzionamento 1968
Interruzioni hardware 1971

Gestione del temporizzatore 1971
Gestione della tastiera 1972
Verifica del funzionamento 1975
Una specie di «shell» 1977
Realizzazione della shell 1977
Conclusione 1978

In questa sezione viene descritto il procedimento per realizzare un sistema, estremamente banalizzato, per elaboratori x86-32, sviluppato prima di os16 e di os32. Questo sistema è privo di pianificazione dei processi (*scheduler*), ma soprattutto non è in grado di avviare programmi e nemmeno di accedere a qualche file system. Tuttavia, pur con tutte le sue mancanze, questa specie di sistema può essere utile per comprendere alcuni concetti a chi inizia da zero lo studio delle problematiche connesse con i sistemi operativi.

Giusto per dare un nome a questa cosa, si usa la sigla «05», ovvero le cifre numeriche che più si avvicinano a «os».

Preparazione

File-immagine 1903

Prima di cominciare conviene preparare tutto quello che serve, come viene descritto nelle sezioni successive. Naturalmente ci si avvale degli strumenti di un sistema GNU/Linux per lo sviluppo di questo giocattolo.

File-immagine

Per prima cosa serve un file-immagine di un dischetto da 1,44 Mibyte, predisposto con GRUB 1, in modo tale da avviare il file ‘`kernel`’. In pratica si predispone inizialmente un dischetto reale, con un file system Dos-FAT, si crea la directory ‘`grub/`’ e al suo interno si mettono i file ‘`stage1`’ e ‘`stage2`’ di GRUB 1, assieme al file ‘`menu.lst`’ che può avere semplicemente il contenuto seguente:

```
title kernel
kernel (fd0)/kernel
```

Si mette temporaneamente un file fittizio, denominato ‘`kernel`’, nella directory principale del dischetto e si procede all’installazione del settore di avvio di GRUB 1 stesso:

```
# grub [Invio]

grub> root (fd0) [Invio]

   Filesystem type is fat, using whole disk.

grub> setup (fd0) [Invio]

   Checking if "/boot/grub/stage1" exists... no
   Checking if "/grub/stage1" exists... yes
   Checking if "/grub/stage2" exists... yes
   Checking if "/grub/fat_stage1_5" exists... no
   Running "install /grub/stage1 (fd0) /grub/stage2 p
     /grub/menu.lst "... succeeded
   Done.

grub> quit [Invio]
```

A questo punto, avendo terminato il lavoro di installazione di GRUB 1 nel dischetto, si può produrre il file-immagine:

```
# cp /dev/fd0 floppy.img [Invio]
```

Directory di lavoro

«

Directory «05/» 1905
Script di collegamento 1907
Altre directory 1908

`bochs` 1905 `compile` 1905 `linker.ld` 1907 `makeit` 1905 `mount` 1905 `umount` 1905

Prima di iniziare gli esperimenti, si predispone una directory di lavoro, da utilizzare in qualità di utente comune. Nella directory si copia il file '`floppy.img`' e si mettono alcuni script molto semplici:

Listato u164.1. '`./mount`'

```
#!/bin/sh
chmod a+rw floppy.img
su root -c "mount -o loop,uid=1001 -t vfat floppy.img /mnt/fd0"
```

Listato u164.2. '`./umount`'

```
#!/bin/sh
su root -c "umount /mnt/fd0"
```

Listato u164.3. '`./bochs`'

```
#!/bin/sh
bochs -q 'boot:a' 'floppya: 1_44=floppy.img, status=inserted' 'megs:32'
```

Il senso di questi script è evidente e il loro scopo è solo quello di ridurre al minimo l'impegno di digitazione. In questa directory viene poi predisposto anche lo script '**`compile`**', ma viene descritto nella sezione successiva.

Directory «05/»

«

A partire dalla directory di lavoro si crea la sottodirectory '`05/`', nella quale viene poi messo il codice del sistema che si va a creare. Ma per evitare di fare confusione con i file-make, si predispone uno script per la compilazione che li crea al volo, in base ai contenuti effettivi delle sottodirectory.

Listato u164.4. '`./05/makeit`'

```
#!/bin/sh
#
# makeit...
#
OPTION="$1"
#
edition () {
    local EDITION="include/kernel/build.h"
    echo -n                                 > $EDITION
    echo -n "#define BUILD_DATE \""     >> $EDITION
    echo -n 'date "+%Y%m%d%H%M%S"'      >> $EDITION
    echo  "\""  >> $EDITION
}
#
#
#
makefile () {
    #
    local MAKEFILE="Makefile"
    local TAB=" "
    #
    local SOURCE_C=""
    local C=""
    local SOURCE_S=""
    local S=""
    #
    local c
    local s
    #
    # Trova i file in C.
    #
    for c in *.c
    do
        if [ -f $c ]
        then
        C='basename $c .c'
            SOURCE_C="$SOURCE_C $C"
        fi
    done
    #
    # Trova i file in ASM.
    #
    for s in *.s
    do
        if [ -f $s ]
        then
        S='basename $s .s'
            SOURCE_S="$SOURCE_S $S"
        fi
```

```
    done
    #
    # Prepara il file make.
    #
    echo -n                                                 > $MAKEFILE
    echo "# Questo file è stato prodotto automaticamente"  >> $MAKEFILE
    echo "# dallo script \"makeit\", sulla base dei"       >> $MAKEFILE
    echo "# contenuti della directory."                    >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "c = $SOURCE_C"                                   >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "s = $SOURCE_S"                                   >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "all: \$(s) \$(c)"                                >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "clean:"                                          >> $MAKEFILE
    echo "${TAB}@rm *.o 2> /dev/null ; pwd"                >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "\$(s):"                                          >> $MAKEFILE
    echo "${TAB}@echo \$@.s"                               >> $MAKEFILE
    echo "${TAB}@as -o \$@.o \$@.s"                        >> $MAKEFILE
    echo "#"                                               >> $MAKEFILE
    echo "\$(c):"                                          >> $MAKEFILE
    echo "${TAB}@echo \$@.c"                               >> $MAKEFILE
    echo "${TAB}@gcc -Wall -Werror -o \$@.o -c \$@.c" \
         "-nostdinc -nostdlib -nostartfiles -nodefaultlibs" \
        "-I../include -I../../include -I../../../include"  >> $MAKEFILE
    #
}
#
#
#
main () {
    #
    local CURDIR=‘pwd‘
    local OBJECTS
    local d
    local c
    local s
    local o
    #
    edition
    #
    for d in ‘find .‘
    do
        if [ -d "$d" ]
        then
            #
            # Ci sono sorgenti in C o in ASM?
            #
            c=‘echo $d/*.c | sed "s/ .*//"‘
            s=‘echo $d/*.s | sed "s/ .*//"‘
            #
            if [ -f "$c" ] || [ -f "$s" ]
            then
                CURDIR=‘pwd‘
                cd $d
                makefile
                #
                if [ "$OPTION" = "clean" ]
                then
                    make clean
                else
                    if ! make
                    then
                        cd "$CURDIR"
                        exit
                    fi
                fi
                cd "$CURDIR"
            fi
        fi
    done
    #
    cd "$CURDIR"
    #
    #
    #
    if [ "$OPTION" = "clean" ]
    then
        true
    else
        OBJECTS=""
        #
        for o in ‘find . -name \*.o -print‘
        do
            if    [ "$o" = "./kernel/kernel_boot.o" ] \
               || [ "$o" = "./kernel/kernel_main.o" ] \
               || [ ! -e "$o" ]
            then
                true
            else
                OBJECTS="$OBJECTS $o"
            fi
        done
        #
        echo "Link"
        #
        ld --script=linker.ld -o kernel_image \
```



```
                kernel/kernel_boot.o \
                $OBJECTS \
                kernel/kernel_main.o
        #
        cp -f kernel_image /mnt/fd0/kernel
        sync
    fi
}
#
# Start.
#
if [ -d include ] && [ -d kernel ] && [ -d lib ]
then
    main
else
    echo "Mi trovo in una posizione sbagliata e non posso svolgere" \
         "il mio compito"
fi
```

Va osservato che la variabile ‘**TAB**’ deve contenere esattamente una tabulazione orizzontale (di norma il codice 09_{16}. Pertanto, se si riproduce il file o se lo si scarica, occorre verificare che il contenuto sia effettivamente una tabulazione, altrimenti va corretto. Se la variabile ‘**TAB**’ contiene solo spazi, i file-make che si ottengono non sono validi.

```
local TAB=" "
```

In pratica, attraverso questo script, i file-make che si generano hanno un aspetto simile a quello del listato seguente:

```
c =  elenco_file_c_senza_estensione
#
s =  elenco_file_asm_senza_estensione
#
all: $(s) $(c)
#
clean:
        @rm *.o 2> /dev/null ; pwd
#
$(s):
        @echo $@.s
        @as -o $@.o $@.s
#
$(c):
        @echo $@.c
        @gcc -Wall -Werror -o $@.o -c $@.c ↩
↪               -nostdinc -nostdlib -nostartfiles ↩
↪               -nodefaultlibs -I../include ↩
↪               -I../../include -I../../../include
```

Il «collegamento» (*link*) dei file avviene attraverso un comando contenuto nello script ‘**makeit**’, dove si fa in modo di mettere all’inizio il file-oggetto che è responsabile dell’avvio, dal momento che contiene l’impronta di riconoscimento per il sistema di avvio aderente alle specifiche *multiboot*.

Nella directory di lavoro descritta nella sezione precedente, conviene mettere uno script che richiami a sua volta ‘**makeit**’ e che provveda a copiare il file del kernel nel file-immagine del dischetto:

Listato u164.6. ‘`./compile`’

```
#!/bin/sh
cd 05
./makeit clean
./makeit
cd ..
```

Script di collegamento

Sempre all’interno della directory ‘`05/`’ va predisposto lo script usato da GNU LD per eseguire correttamente il collegamento dei file oggetto in un file eseguibile unico. Dal momento che nel progetto che si intraprende si intende usare la memoria linearmente, si intende che il blocco minimo sia della dimensione di un registro, ovvero pari a 4 byte:

Listato u164.7. ‘./05/linker.ld’

```
/*****************************************************************************
 * La memoria viene usata in modo lineare, senza controlli dei
 * privilegi, così non si usano nemmeno gli allineamenti tradizionali
 * di 4096 byte, ma solo di 4 byte, ovvero di un registro.
 *****************************************************************************/

ENTRY (kernel_boot)
SECTIONS {
    . = 0x00100000;
    k_mem_total_s = .;
    .text : {
        k_mem_text_s = .;
        *(.text)
        . = ALIGN (0x4);
        k_mem_text_e = .;
    }
    .rodata : {
        k_mem_rodata_s = .;
        *(.rodata)
        . = ALIGN (0x4);
        k_mem_rodata_e = .;
    }
    .data : {
        k_mem_data_s = .;
        *(.data)
        . = ALIGN (0x4);
        k_mem_data_e = .;
    }
    .bss : {
        k_mem_bss_s = .;
        *(.bss)
        *(COMMON)
        . = ALIGN (0x4);
        k_mem_bss_e = .;
    }
    k_mem_total_e = .;
}
```

Il codice contenuto nel file del kernel che si va a produrre, deve iniziare a partire da 00100000_{16}, ovvero da 1 Mibyte, come prescrive il sistema di avvio *multiboot*, il quale va a collocarlo in memoria, a partire da quella posizione. Inoltre, per consentire di individuare i blocchi di memoria utilizzati, vengono inseriti dei simboli; per esempio, ‘**k_mem_total_s**’ individua l’inizio del kernel, mentre ‘**k_mem_total_e**’ ne individua la fine.

Si dà per scontato che GNU AS predisponga un file eseguibile in formato ELF.

Altre directory

All’interno di ‘05/’ si creano ancora: ‘lib/’, per la libreria standard e altre librerie specifiche del sistema; ‘include/’, per i file di intestazione della libreria; ‘kernel/’ con i file iniziali usati dal kernel; ‘app/’ per le applicazioni (ovvero le funzioni avviate dal kernel quando tutto è pronto).

Libreria standard per iniziare

Libreria «limits.h» 1909

limits.h 1909

Quando si scrive un programma da utilizzare senza l’ausilio del sistema operativo, è necessario realizzare una propria libreria di funzioni C, perché quella che offre il proprio compilatore, è fatta sicuramente per interagire con il sistema operativo che la ospita. Nelle sezioni successive vengono mostrati i file usati nel sistema in corso di presentazione, per una libreria C standard generalizzata.

Va però osservato che possono essere gestiti solo interi con un massimo di 32 bit. Infatti, il compilatore GNU C consentirebbe anche di gestire interi a 64 bit, corrispondenti al tipo ‘**long long**’, ma per questo si avvale di funzioni di libreria non standard che, però, qui non sono state realizzate.

Libreria «limits.h»

Il file ‘limits.h’ dimostra quanto appena accennato a proposito della limitazione nella gestione dei numeri interi. Contrariamente a quanto si fa di solito, i valori sono scritti in esadecimale.

Listato u165.1. ‘./05/include/limits.h’

```
#ifndef _LIMITS_H
#define _LIMITS_H       1

#define CHAR_BIT            (8)
#define SCHAR_MIN       (-0x80)
#define SCHAR_MAX        (0x7F)
#define UCHAR_MAX        (0xFF)
#define CHAR_MIN     SCHAR_MIN
#define CHAR_MAX     SCHAR_MAX
#define MB_LEN_MAX          (16)
#define SHRT_MIN      (-0x8000)
#define SHRT_MAX       (0x7FFF)
#define USHRT_MAX      (0xFFFF)
#define INT_MIN   (-0x80000000)
#define INT_MAX    (0x7FFFFFFF)
#define UINT_MAX   (0xFFFFFFFFU)
#define LONG_MIN  (-0x80000000L)
#define LONG_MAX   (0x7FFFFFFFL)
#define ULONG_MAX  (0xFFFFFFFFUL)

#endif
```

Libreria «stdbool.h» 1911
Libreria «time.h» 1912
Libreria «ctype.h» 1912
Libreria «stdint.h» 1913
Libreria «inttypes.h» 1914
Libreria «stdarg.h» 1917
Libreria «stddef.h» 1917
Libreria «stdlib.h» 1917
Libreria «string.h» 1918
Libreria «stdio.h» 1920

`atoi.c` 1917 `ctype.h` 1912 `inttypes.h` 1914 `memcpy.c` 1918 `memset.c` 1918 `NULL.h` 1911 `ptrdiff_t.h` 1911 `restrict.h` 1911 `size_t.h` 1911 `snprintf.c` 1920 `stdarg.h` 1917 `stdbool.h` 1911 `stddef.h` 1917 `stdint.h` 1913 `stdio.h` 1920 `stdlib.h` 1917 `string.h` 1918 `strncpy.c` 1918 `time.h` 1912 `vsnprintf.c` 1920 `wchar_t.h` 1911

Secondo lo standard, più file di libreria dichiarano gli stessi tipi speciali e le stesse costanti. Per evitare confusione, la dichiarazione di queste costanti e di questi tipi condivisi, viene collocata in file isolati che, successivamente, altri file incorporano a seconda della necessità. Inoltre, il compilatore usato per la costruzione di questo sistema non gestisce i «puntatori ristretti», ovvero non considera valida la parola chiave '**`restrict`**'. Per mantenere una forma aderente allo standard si aggiunge la dichiarazione della macro-variabile ***restrict*** vuota, in un file separato che molti altri file incorporano.

Listato u166.1. '`./05/include/restrict.h`'

```
#ifndef _RESTRICT_H
#define _RESTRICT_H     1

#define restrict

#endif
```

Listato u166.2. '`./05/include/NULL.h`'

```
#ifndef _NULL_H
#define _NULL_H     1

#define NULL 0

#endif
```

Listato u166.3. '`./05/include/ptrdiff_t.h`'

```
#ifndef _PTRDIFF_T_H
#define _PTRDIFF_T_H    1

typedef long int ptrdiff_t;

#endif
```

Listato u166.4. '`./05/include/size_t.h`'

```
#ifndef _SIZE_T_H
#define _SIZE_T_H      1

typedef unsigned long int size_t;

#endif
```

Listato u166.5. '`./05/include/wchar_t.h`'

```
#ifndef _WCHAR_T_H
#define _WCHAR_T_H     1

typedef unsigned char wchar_t;

#endif
```

Dal file '`wchar_t.h`' si comprende che, per il sistema in corso di realizzazione, si intende gestire al massimo la codifica ASCII e nulla di più.

Libreria «stdbool.h»

Listato u166.6. '`./05/include/stdbool.h`'

```
#ifndef _STDBOOL_H
#define _STDBOOL_H      1

#define bool    _Bool
#define true    1
#define false   0
#define __bool_true_false_are_defined   1

#endif
```

Libreria «time.h»

Listato u166.7. '`./05/include/time.h`'

```
#ifndef _TIME_H
#define _TIME_H       1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

#define CLOCKS_PER_SEC  100L
typedef long int clock_t;
typedef long int time_t;

struct tm {int tm_sec;  int tm_min;  int tm_hour;
           int tm_mday; int tm_mon;  int tm_year;
           int tm_wday; int tm_yday; int tm_isdst;};

clock_t    clock     (void);
time_t     time      (time_t *timer);
double     difftime  (time_t time1, time_t time0);
time_t     mktime    (struct tm *timeptr);
struct tm *gmtime    (const time_t *timer);
struct tm *localtime (const time_t *timer);
char      *asctime   (const struct tm *timeptr);
char      *ctime     (const time_t *timer);
size_t    strftime   (char * restrict s, size_t maxsize,
                      const char * restrict format,
                      const struct tm * restrict timeptr);

#define ctime(t) (asctime (localtime (t)));

#endif
```

Del file '`time.h`' viene usato solo il tipo '**`clock_t`**' e la macro-variabile ***CLOCKS_PER_SEC***, con la quale si dichiara implicitamente la frequenza con cui deve reagire il temporizzatore interno del realizzando sistema. Pertanto, le funzioni del file di cui si vedono i prototipi, non vengono realizzate.

Libreria «ctype.h»

Listato u166.8. '`./05/include/ctype.h`'

```
#ifndef _CTYPE_H
#define _CTYPE_H       1

#include <NULL.h>

#define isblank(C)  ((int) (C == ' ' || C == '\t'))
#define isspace(C)  ((int) (C == ' ' \
                            || C == '\f' \
                            || C == '\n' \
                            || C == '\n' \
                            || C == '\r' \
                            || C == '\t' \
                            || C == '\v'))
#define isdigit(C)  ((int) (C >= '0' && C <= '9'))
#define isxdigit(C) ((int) ((C >= '0' && C <= '9') \
                             || (C >= 'A' && C <= 'F') \
                             || (C >= 'a' && C <= 'f')))
#define isupper(C)  ((int) (C >= 'A' && C <= 'Z'))
#define islower(C)  ((int) (C >= 'a' && C <= 'z'))
#define iscntrl(C)  \
    ((int) ((C >= 0x00 && C <= 0x1F) || C == 0x7F))
#define isgraph(C)  ((int) (C >= 0x21 && C <= 0x7E))
#define isprint(C)  ((int) (C >= 0x20 && C <= 0x7E))
#define isalpha(C)  (isupper (C) || islower (C))
```



```
#define isalnum(C)  (isalpha (C) || isdigit (C))
#define ispunct(C)  \
    (isgraph (C) && (!isspace (C)) && (!isalnum (C)))

#endif
```

Libreria «stdint.h»

Listato u166.9. '`./05/include/stdint.h`'

```
#ifndef _STDINT_H
#define _STDINT_H       1

typedef signed char                  int8_t;
typedef short int                   int16_t;
typedef int                         int32_t;      // x86-32
typedef unsigned char               uint8_t;
typedef unsigned short int         uint16_t;
typedef unsigned int               uint32_t;      // x86-32

#define INT8_MIN                     (-0x80)
#define INT8_MAX                      (0x7F)
#define UINT8_MAX                     (0xFF)
#define INT16_MIN                   (-0x8000)
#define INT16_MAX                    (0x7FFF)
#define UINT16_MAX                   (0xFFFF)
#define INT32_MIN               (-0x80000000)
#define INT32_MAX                (0x7FFFFFFF)
#define UINT32_MAX               (0xFFFFFFFFU)

typedef signed char             int_least8_t;
typedef short int              int_least16_t;
typedef int                    int_least32_t;
typedef unsigned char          uint_least8_t;
typedef unsigned short int    uint_least16_t;
typedef unsigned int          uint_least32_t;

#define INT_LEAST8_MIN                (-0x80)
#define INT_LEAST8_MAX                 (0x7F)
#define UINT_LEAST8_MAX                (0xFF)
#define INT_LEAST16_MIN              (-0x8000)
#define INT_LEAST16_MAX               (0x7FFF)
#define UINT_LEAST16_MAX              (0xFFFF)
#define INT_LEAST32_MIN         (-0x80000000)
#define INT_LEAST32_MAX          (0x7FFFFFFF)
#define UINT_LEAST32_MAX        (0xFFFFFFFFU)

#define INT8_C(VAL)     VAL
#define INT16_C(VAL)    VAL
#define INT32_C(VAL)    VAL
#define UINT8_C(VAL)    VAL
#define UINT16_C(VAL)   VAL
#define UINT32_C(VAL)   VAL ## U

typedef signed char              int_fast8_t;
typedef int                     int_fast16_t;
typedef int                     int_fast32_t;
typedef unsigned char           uint_fast8_t;
typedef unsigned int           uint_fast16_t;
typedef unsigned int           uint_fast32_t;

#define INT_FAST8_MIN                 (-0x80)
#define INT_FAST8_MAX                  (0x7F)
#define UINT_FAST8_MAX                 (0xFF)
#define INT_FAST16_MIN          (-0x80000000)
#define INT_FAST16_MAX           (0x7FFFFFFF)
#define UINT_FAST16_MAX          (0xFFFFFFFFU)
#define INT_FAST32_MIN          (-0x80000000)
#define INT_FAST32_MAX           (0x7FFFFFFF)
#define UINT_FAST32_MAX          (0xFFFFFFFFU)

typedef int                         intptr_t;
typedef unsigned int               uintptr_t;

#define INTPTR_MIN              (-0x80000000)
#define INTPTR_MAX              (0x7FFFFFFF)
#define UINTPTR_MAX             (0xFFFFFFFFU)

typedef long int                    intmax_t;
typedef unsigned long int          uintmax_t;

#define INTMAX_C(VAL)   VAL ## L
```

```
#define UINTMAX_C(VAL)  VAL ## UL

#define INTMAX_MIN             (-0x80000000L)
#define INTMAX_MAX              (0x7FFFFFFFL)
#define UINTMAX_MAX             (0xFFFFFFFFUL)

#define PTRDIFF_MIN            (-0x80000000)
#define PTRDIFF_MAX             (0x7FFFFFFF)

#define SIG_ATOMIC_MIN         (-0x80000000)
#define SIG_ATOMIC_MAX          (0x7FFFFFFF)

#define SIZE_MAX                (0xFFFFFFFFU)

#define WCHAR_MIN                       (0)
#define WCHAR_MAX                   (0xFFFFU)

#define WINT_MIN                   (-0x8000L)
#define WINT_MAX                    (0x7FFFL)

#endif
```

Libreria «inttypes.h»

«

Listato u166.10. ‘./05/include/inttypes.h’

```
#ifndef _INTTYPES_H
#define _INTTYPES_H     1

#include <restrict.h>
#include <stdint.h>
#include <wchar_t.h>

typedef struct {intmax_t quot; intmax_t rem;} imaxdiv_t;

#define PRId8           "d"
#define PRId16          "d"
#define PRId32          "d"
#define PRId64          "lld"
#define PRIdLEAST8      "d"
#define PRIdLEAST16     "d"
#define PRIdLEAST32     "d"
#define PRIdLEAST64     "lld"
#define PRIdFAST8       "d"
#define PRIdFAST16      "d"
#define PRIdFAST32      "d"
#define PRIdFAST64      "lld"
#define PRIdMAX         "lld"
#define PRIdPTR         "d"
#define PRIi8           "i"
#define PRIi16          "i"
#define PRIi32          "i"
#define PRIi64          "lli"
#define PRIiLEAST8      "i"
#define PRIiLEAST16     "i"
#define PRIiLEAST32     "i"
#define PRIiLEAST64     "lli"
#define PRIiFAST8       "i"
#define PRIiFAST16      "i"
#define PRIiFAST32      "i"
#define PRIiFAST64      "lli"
#define PRIiMAX         "lli"
#define PRIiPTR         "i"
#define PRIb8           "b"     // PRIb... non è standard
#define PRIb16          "b"     //
#define PRIb32          "b"     //
#define PRIb64          "llb"   //
#define PRIbLEAST8      "b"     //
#define PRIbLEAST16     "b"     //
#define PRIbLEAST32     "b"     //
#define PRIbLEAST64     "llb"   //
#define PRIbFAST8       "b"     //
#define PRIbFAST16      "b"     //
#define PRIbFAST32      "b"     //
#define PRIbFAST64      "llb"   //
#define PRIbMAX         "llb"   //
#define PRIbPTR         "b"     //
#define PRIo8           "o"
#define PRIo16          "o"
#define PRIo32          "o"
#define PRIo64          "llo"
```



```
#define PRIoLEAST8      "o"
#define PRIoLEAST16     "o"
#define PRIoLEAST32     "o"
#define PRIoLEAST64     "llo"
#define PRIoFAST8       "o"
#define PRIoFAST16      "o"
#define PRIoFAST32      "o"
#define PRIoFAST64      "llo"
#define PRIoMAX         "llo"
#define PRIoPTR         "o"
#define PRIu8           "u"
#define PRIu16          "u"
#define PRIu32          "u"
#define PRIu64          "llu"
#define PRIuLEAST8      "u"
#define PRIuLEAST16     "u"
#define PRIuLEAST32     "u"
#define PRIuLEAST64     "llu"
#define PRIuFAST8       "u"
#define PRIuFAST16      "u"
#define PRIuFAST32      "u"
#define PRIuFAST64      "llu"
#define PRIuMAX         "llu"
#define PRIuPTR         "u"
#define PRIx8           "x"
#define PRIx16          "x"
#define PRIx32          "x"
#define PRIx64          "llx"
#define PRIxLEAST8      "x"
#define PRIxLEAST16     "x"
#define PRIxLEAST32     "x"
#define PRIxLEAST64     "llx"
#define PRIxFAST8       "x"
#define PRIxFAST16      "x"
#define PRIxFAST32      "x"
#define PRIxFAST64      "llx"
#define PRIxMAX         "llx"
#define PRIxPTR         "x"
#define PRIX8           "X"
#define PRIX16          "X"
#define PRIX32          "X"
#define PRIX64          "llX"
#define PRIXLEAST8      "X"
#define PRIXLEAST16     "X"
#define PRIXLEAST32     "X"
#define PRIXLEAST64     "llX"
#define PRIXFAST8       "X"
#define PRIXFAST16      "X"
#define PRIXFAST32      "X"
#define PRIXFAST64      "llX"
#define PRIXMAX         "llX"
#define PRIXPTR         "X"

#define SCNd8           "hhd"
#define SCNd16          "hd"
#define SCNd32          "d"
#define SCNd64          "lld"
#define SCNdLEAST8      "hhd"
#define SCNdLEAST16     "hd"
#define SCNdLEAST32     "d"
#define SCNdLEAST64     "lld"
#define SCNdFAST8       "hhd"
#define SCNdFAST16      "d"
#define SCNdFAST32      "d"
#define SCNdFAST64      "lld"
#define SCNdMAX         "lld"
#define SCNdPTR         "d"
#define SCNi8           "hhi"
#define SCNi16          "hi"
#define SCNi32          "i"
#define SCNi64          "lli"
#define SCNiLEAST8      "hhi"
#define SCNiLEAST16     "hi"
#define SCNiLEAST32     "i"
#define SCNiLEAST64     "lli"
#define SCNiFAST8       "hhi"
#define SCNiFAST16      "i"
#define SCNiFAST32      "i"
#define SCNiFAST64      "lli"
#define SCNiMAX         "lli"
#define SCNiPTR         "i"
```

```
#define SCNb8           "hhb"   // SCNb... non è standard
#define SCNb16          "hb"    //
#define SCNb32          "b"     //
#define SCNb64          "llb"   //
#define SCNbLEAST8      "hhb"   //
#define SCNbLEAST16     "hb"    //
#define SCNbLEAST32     "b"     //
#define SCNbLEAST64     "llb"   //
#define SCNbFAST8       "hhb"   //
#define SCNbFAST16      "b"     //
#define SCNbFAST32      "b"     //
#define SCNbFAST64      "llb"   //
#define SCNbMAX         "llb"   //
#define SCNbPTR         "b"     //
#define SCNo8           "hho"
#define SCNo16          "ho"
#define SCNo32          "o"
#define SCNo64          "llo"
#define SCNoLEAST8      "hho"
#define SCNoLEAST16     "ho"
#define SCNoLEAST32     "o"
#define SCNoLEAST64     "llo"
#define SCNoFAST8       "hho"
#define SCNoFAST16      "o"
#define SCNoFAST32      "o"
#define SCNoFAST64      "llo"
#define SCNoMAX         "llo"
#define SCNoPTR         "o"
#define SCNu8           "hhu"
#define SCNu16          "hu"
#define SCNu32          "u"
#define SCNu64          "llu"
#define SCNuLEAST8      "hhu"
#define SCNuLEAST16     "hu"
#define SCNuLEAST32     "u"
#define SCNuLEAST64     "llu"
#define SCNuFAST8       "hhu"
#define SCNuFAST16      "u"
#define SCNuFAST32      "u"
#define SCNuFAST64      "llu"
#define SCNuMAX         "llu"
#define SCNuPTR         "u"
#define SCNx8           "hhx"
#define SCNx16          "hx"
#define SCNx32          "x"
#define SCNx64          "llx"
#define SCNxLEAST8      "hhx"
#define SCNxLEAST16     "hx"
#define SCNxLEAST32     "x"
#define SCNxLEAST64     "llx"
#define SCNxFAST8       "hhx"
#define SCNxFAST16      "x"
#define SCNxFAST32      "x"
#define SCNxFAST64      "llx"
#define SCNxMAX         "llx"
#define SCNxPTR         "x"

imaxdiv_t imaxdiv    (intmax_t numer, intmax_t denom);
intmax_t  strtoimax  (const char *restrict nptr,
                      char **restrict endptr,
                      int base);
uintmax_t strtouimax (const char *restrict nptr,
                      char **restrict endptr,
                      int base);
intmax_t  wcstoimax  (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr,
                      int base);
uintmax_t wcstouimax (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr,
                      int base);
#endif
```

La libreria ‘`inttypes.h`’ serve per le macro-variabili del tipo ‘**`PRI`*`xn`***’, in modo da utilizzare correttamente la funzione ***printf()***, mentre si fa riferimento a tipi di valori numerici definiti nel file ‘`stdint.h`’. Pertanto, le funzioni non vengono realizzate.

Libreria «stdarg.h»

Listato u166.11. ‘`./05/include/stdarg.h`’

```
#ifndef _STDARG_H
#define _STDARG_H       1

typedef unsigned char *va_list;

#define va_start(ap, last) ((void) ((ap) = \
    ((va_list) &(last)) + (sizeof (last))))
#define va_end(ap)         ((void) ((ap) = 0))
#define va_copy(dest, src) \
    ((void) ((dest) = (va_list) (src)))
#define va_arg(ap, type)   \
    (((ap) = (ap) + (sizeof (type))), \
        *((type *) ((ap) - (sizeof (type)))))
#endif
```

Libreria «stddef.h»

Listato u166.12. ‘`./05/include/stddef.h`’

```
#ifndef _STDDEF_H
#define _STDDEF_H      1

#include <ptrdiff_t.h>
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>

#define offsetof(type, member)  \
    ((size_t) &((type *)0)->member)

#endif
```

Libreria «stdlib.h»

Listato u166.13. ‘`./05/include/stdlib.h`’

```
#ifndef _STDLIB_H
#define _STDLIB_H      1

#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>
#include <limits.h>
#include <restrict.h>

typedef struct {int quot; int rem;} div_t;
typedef struct {long int quot; long int rem;} ldiv_t;
typedef struct {long long int quot; long long int rem;} lldiv_t;

#define EXIT_FAILURE    1
#define EXIT_SUCCESS    0
#define RAND_MAX        INT_MAX
#define MB_CUR_MAX      ((size_t) MB_LEN_MAX)

int            atoi  (const char *nptr);
long int       atol  (const char *nptr);
long long int  atoll (const char *nptr);
double         atof  (const char *nptr);

float                 strtof   (const char * restrict nptr,
                                char ** restrict endptr);
double                strtod   (const char * restrict nptr,
                                char ** restrict endptr);
long double           strtold  (const char * restrict nptr,
                                char ** restrict endptr);
long int              strtol   (const char * restrict nptr,
                                char ** restrict endptr, int base);
long long int         strtoll  (const char * restrict nptr,
                                char ** restrict endptr, int base);
unsigned long int     strtoul  (const char * restrict nptr,
                                char ** restrict endptr, int base);
unsigned long long int strtoull (const char * restrict nptr,
                                char ** restrict endptr, int base);

int   rand    (void);
void  srand   (unsigned int seed);

void *malloc  (size_t size);
void *realloc (void *ptr, size_t size);
void  free    (void *ptr);
#define calloc(nmemb, size) (malloc ((nmemb) * (size)))

int   atexit  (void (*func) (void));
void  exit    (int status);
void  _Exit   (int status);
void  abort   (void);

char *getenv  (const char *name);
int   system  (const char *string);
```

```
void  qsort   (void *base,
               size_t nmemb,
               size_t size,
               int (*compar) (const void *, const void *));
void *bsearch (const void *key,
               const void *base,
               size_t nmemb,
               size_t size,
               int (*compar) (const void *, const void *));

int abs             (int j);
long int labs       (long int j);
long long int llabs (long long int j);

div_t   div   (int numer, int denom);
ldiv_t  ldiv  (long int numer, long int denom);
lldiv_t lldiv (long long int numer, long long int denom);

int    mblen    (const char *s, size_t n);
int    mbtowc   (wchar_t *restrict pwc, const char *restrict s, size_t n);
int    wctomb   (char *s, wchar_t wc);
size_t mbstowcs (wchar_t *restrict pwcs, const char *restrict s, size_t n);
size_t wcstombs (char *restrict s, const wchar_t *restrict pwcs, size_t n);

#endif
```

Di questa libreria vengono realizzate solo alcune funzioni, ma in particolare, ***_Exit()***, ***malloc()***, ***realloc()*** e ***free()***, dipendono strettamente dal contesto del sistema; pertanto vengono mostrate a parte, in un'altra sezione più specifica.

Listato u166.14. '`./05/lib/atoi.c`'

```
#include <stdlib.h>
#include <ctype.h>
int
atoi (const char *nptr)
{
    int i;
    int sign = +1;
    int n;

    for (i = 0; isspace (nptr[i]); i++)
      {
        ;       // Si limita a saltare gli spazi iniziali.
      }

    if (nptr[i] == '+')
      {
        sign = +1;
        i++;
      }
    else if (nptr[i] == '-')
      {
        sign = -1;
        i++;
      }

    for (n = 0; isdigit (nptr[i]); i++)
      {
        // Accumula il valore.
        n = (n * 10) + (nptr[i] - '0');
      }

    return sign * n;
}
```

Libreria «string.h»

Listato u166.15. '`./05/include/string.h`'

```
#ifndef _STRING_H
#define _STRING_H       1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

void *memcpy  (void *restrict dst, const void *restrict org,
               size_t n);
void *memmove (void *dst, const void *org, size_t n);

char *strcpy  (char *restrict dst,
               const char *restrict org);
char *strncpy (char *restrict dst, const char *restrict org,
               size_t n);
```



```
char *strcat  (char *restrict dst, const char *restrict org);
char *strncat (char *restrict dst, const char *restrict org,
               size_t n);

int    memcmp  (const void *s1, const void *s2, size_t n);
int    strcmp  (const char *s1, const char *s2);
int    strcoll (const char *s1, const char *s2);
int    strncmp (const char *s1, const char *s2, size_t n);
size_t strxfrm (char *restrict dst,
                const char *restrict org, size_t n);

void  *memchr  (const void *s, int c, size_t n);
char  *strchr  (const char *s, int c);
char  *strrchr (const char *s, int c);
size_t strspn  (const char *s, const char *accept);
size_t strcspn (const char *s, const char *reject);
char  *strpbrk (const char *s, const char *accept);
char  *strstr  (const char *string, const char *substring);
char  *strtok  (char *restrict string,
                const char *restrict delim);

void  *memset   (void *s, int c, size_t n);
char  *strerror (int errnum);
size_t strlen   (const char *s);

#endif
```

Delle funzioni dichiarate nel file '`string.h`' vengono realizzate solo quelle dei listati successivi.

Listato u166.16. '`./05/lib/memset.c`'

```
#include <string.h>
void
*memset (void *s, int c, size_t n)
{
    unsigned char *a = (unsigned char *) s;
    unsigned char x  = (unsigned char) c;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
      {
        a[i] = x;
      }
    return s;
}
```

Listato u166.17. '`./05/lib/strncpy.c`'

```
#include <string.h>
char
*strncpy (char *restrict dst, const char *restrict org,
          size_t n)
{
    size_t i;
    for (i = 0; n > 0 && i < n && org[i] != 0; i++)
      {
        dst[i] = org[i];
      }
    for ( ; n > 0 && i < n; i++)
      {
        dst[i] = 0;
      }
    return dst;
}
```

Listato u166.18. '`./05/lib/memcpy.c`'

```
#include <string.h>
void *
memcpy (void *restrict dst, const void *restrict org,
        size_t n)
{
    unsigned char *d = (unsigned char *) dst;
    unsigned char *o = (unsigned char *) org;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
      {
        d[i] = o[i];
      }
    return dst;
}
```

Libreria «stdio.h»

«

La libreria che è rappresentata dal file ‘`stdio.h`’ è la più noiosa di questo gruppo iniziale. Qui viene mostrato un file incompleto, contenente solo ciò che serve al sistema in corso di realizzazione.

Listato u166.19. ‘`./05/include/stdio.h`’

```
#ifndef _STDIO_H
#define _STDIO_H        1

#include <restrict.h>
#include <size_t.h>
#include <stdarg.h>
#include <stdint.h>
#include <kernel/vga.h>

int vsnprintf (char *restrict s, size_t n,
               const char *restrict format, va_list arg);
int snprintf  (char *restrict s, size_t n,
               const char *restrict format, ...);

#define vsnprintf(s, n, format, arg) (vsnprintf (s, n, format, arg))
#define vsprintf(s, format, arg)     (vsnprintf (s, SIZE_MAX, format, arg))
#define sprintf(s, ...)              (snprintf (s, SIZE_MAX, __VA_ARGS__))

#define vprintf(format, arg)         (vga_vprintf (format, arg))
#define printf(...)                  (vga_printf (__VA_ARGS__))
#define puts(s)                      (vga_puts (s, SIZE_MAX) ; \
                                      vga_puts ("\n", 2))
#define putchar(c)                   (vga_putc (c), c)

char *gets (char *s);

// Il resto del file «stdio.h» standard viene omesso.
#endif
```

Le uniche funzioni che si possono realizzare in modo generalizzato sono ***vsnprintf()*** e ***snprintf()***; tuttavia, la realizzazione che viene mostrata è incompleta, in quanto si consente solo la visualizzazione di numeri interi e stringhe. Nel listato successivo, relativo a ‘`vsnprintf.c`’, si vedono diverse funzioni dichiarate in modo «statico», dato che servono esclusivamente a ***vsnprintf()***.

Listato u166.20. ‘`./05/lib/vsnprintf.c`’

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
//
// Converte un intero senza segno di rango massimo in una stringa.
//
static size_t
uimaxtoa (uintmax_t integer, char *buffer, int base, int uppercase, size_t n)
{
        uintmax_t  integer_copy = integer;
           size_t  digits;
              int  b;
    unsigned char  remainder;

    for (digits = 0; integer_copy > 0; digits++)
      {
        integer_copy = integer_copy / base;
      }

    if (buffer == NULL && integer == 0) return 1;
    if (buffer == NULL && integer > 0)  return digits;

    if (integer == 0)
      {
        buffer[0] = '0';
        buffer[1] = '\0';
        return 1;
      }

    if (n > 0 && digits > n) digits = n;  // Sistema il numero massimo
                                          // di cifre.

    *(buffer + digits) = '\0';            // Fine della stringa.

    for (b = digits - 1; integer != 0 && b >= 0; b--)
      {
        remainder = integer % base;
        integer   = integer / base;

        if (remainder <= 9)
          {
            *(buffer + b) = remainder + '0';
          }
        else
          {
            if (uppercase)
              {
                *(buffer + b) = remainder - 10 + 'A';
              }
            else
              {
                *(buffer + b) = remainder - 10 + 'a';
              }
          }
      }
    return digits;
}
//
// Converte un intero con segno di rango massimo in una stringa.
//
static size_t
imaxtoa (intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, n);
      }
    //
    // A questo punto c'è un valore negativo, inferiore a zero.
    //
    if (buffer == NULL)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, n) + 1;
      }

    *buffer = '-';              // Serve il segno meno all'inizio.
    if (n == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, n-1) + 1;
      }
}
//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo.
//
static size_t
simaxtoa (intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{

    if (buffer == NULL && integer >= 0)
      {
        return uimaxtoa (integer, NULL, base, uppercase, n) + 1;
      }

    if (buffer == NULL && integer < 0)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, n) + 1;
      }
    //
    // A questo punto «buffer» è diverso da NULL.
    //
    if (integer >= 0)
      {
        *buffer = '+';
      }
    else
      {
        *buffer = '-';
      }

    if (n == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer+1, base, uppercase, n-1) + 1;
      }
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, n-1) + 1;
      }
}
//
// Converte un intero senza segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
uimaxtoa_fill (uintmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = uimaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
```

```
      {
        return uimaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        uimaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        uimaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
imaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = imaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    // size_i <= abs (width).

    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        imaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        imaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo, provvedendo a sistemare
// l'allineamento.
//
static size_t
simaxtoa_fill (intmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = simaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
```



```
      {
        return simaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        simaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        simaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Trasferisce una stringa provvedendo all'allineamento.
//
static size_t
strtostr_fill (char *string, char *buffer, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_s = strlen (string);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (width != 0 && size_s > abs (width))
      {
        memcpy (buffer, string, abs (width));
        buffer[width] = '\0';
        return width;
      }

    if (width == 0 && max > 0 && size_s > max)
      {
        memcpy (buffer, string, max);
        buffer[max] = '\0';
        return max;
      }

    if (width == 0 && max > 0 && size_s < max)
      {
        memcpy (buffer, string, size_s);
        buffer[size_s] = '\0';
        return size_s;
      }
    //
    // width =! 0
    // size_s <= abs (width)
    //
    size_f = abs (width) - size_s;

    if (width < 0)
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        strncpy (buffer+size_f, string, size_s);
      }
    else
      {
        // Allineamento a sinistra.
        strncpy (buffer, string, size_s);
        memset (buffer+size_s, filler, size_f);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// La funzione «vsnprintf()»
//
int
vsnprintf (char *restrict string, size_t n,
           const char *restrict format, va_list ap)
{

    if (n > INT_MAX) n = INT_MAX;       // «n» non può essere superiore
                                        // a INT_MAX.
    //
    // Al massimo si producono "n-1" caratteri, + '\0'.
    // "n" viene usato anche come dimensione massima per le
    // stringhe interne, se non è troppo grande.
    //
    int        f                = 0;
    int        s                = 0;
    int        remain           = n - 1;
```

```
bool       specifier           = 0;
bool       specifier_flags     = 0;
bool       specifier_width     = 0;
bool       specifier_precision = 0;
bool       specifier_type      = 0;

bool       flag_plus           = 0;
bool       flag_minus          = 0;
bool       flag_space          = 0;
bool       flag_alternate      = 0;
bool       flag_zero           = 0;

int        alignment;
int        filler;

intmax_t   value_i;
uintmax_t  value_ui;
char      *value_cp;

size_t     width;
size_t     precision;
size_t     str_size = n > 1024 ? 1024 : n;
char       width_string[str_size];
char       precision_string[str_size];
int        w;
int        p;

width_string[0]     = '\0';
precision_string[0] = '\0';

while (format[f] != 0 && s < (n - 1))
  {
    if (!specifier)
      {
        if (format[f] != '%')
          {
            string[s] = format[f];
            s++;
            remain--;
            f++;
            continue;
          }
        if (format[f] == '%' && format[f+1] == '%')
          {
            string[s] = '%';
            f++;
            f++;
            s++;
            remain--;
            continue;
          }
        if (format[f] == '%')
          {
            f++;
            specifier = 1;
            specifier_flags = 1;
            continue;
          }
      }

    if (specifier && specifier_flags)
      {
        if (format[f] == '+')
          {
            flag_plus = 1;
            f++;
            continue;
          }
        else if (format[f] == '-')
          {
            flag_minus = 1;
            f++;
            continue;
          }
        else if (format[f] == ' ')
          {
            flag_space = 1;
            f++;
            continue;
          }
        else if (format[f] == '#')
          {
            flag_alternate = 1;
            f++;
            continue;
          }
        else if (format[f] == '0')
          {
            flag_zero = 1;
            f++;
            continue;
          }
        else
          {
            specifier_flags = 0;
            specifier_width = 1;
          }
      }

    if (specifier && specifier_width)
      {
        for (w = 0; format[f] >= '0' && format[f] <= '9'
```



```
             && w < str_size; w++)
          {
            width_string[w] = format[f];
            f++;
          }
        width_string[w] = '\0';

        specifier_width = 0;

        if (format[f] == '.')
          {
            specifier_precision = 1;
            f++;
          }
        else
          {
            specifier_precision = 0;
            specifier_type      = 1;
          }
      }

    if (specifier && specifier_precision)
      {
        for (p = 0; format[f] >= '0' && format[f] <= '9'
             && p < str_size; p++)
          {
            precision_string[p] = format[f];
            p++;
          }
        precision_string[p] = '\0';

        specifier_precision = 0;
        specifier_type      = 1;
      }

    if (specifier && specifier_type)
      {
        width     = atoi (width_string);
        precision = atoi (precision_string);
                    filler = ' ';
        if (flag_zero)  filler = '0';
        if (flag_space) filler = ' ';
                    alignment = width;
        if (flag_minus)
          {
            alignment = -alignment;
            filler = ' ';   // Il carattere di riempimento
                            // non può essere zero.
          }

        if (format[f] == 'h' && format[f+1] == 'h')
          {
            if (format[f+2] == 'd' || format[f+2] == 'i')
              {
                // signed char, base 10.
                value_i = va_arg (ap, int);
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 3;
              }
            else if (format[f+2] == 'u')
              {
                // unsigned char, base 10.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'o')
              {
                // unsigned char, base 8.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'x')
              {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'X')
              {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'b')
              {
                // unsigned char, base 2 (estensione).
```

```
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else // Specificatore errato;
              {
                f += 2;
              }
          }
        else if (format[f] == 'h')
          {
            if (format[f+1] == 'd' || format[f+1] == 'i')
              {
                // short int, base 10.
                value_i = va_arg (ap, int);
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 2;
              }
            else if (format[f+1] == 'u')
              {
                // unsigned short int, base 10.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'o')
              {
                // unsigned short int, base 8.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'x')
              {
                // unsigned short int, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'X')
              {
                // unsigned short int, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'b')
              {
                // unsigned short int, base 2 (estensione).
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else // Specificatore errato;
              {
                f += 1;
              }
          }
        //
        // Il tipo «long long int» non c'è, perché il compilatore
        // GNU C, per poter eseguire le divisioni e il calcolo del
        // resto, ha bisogno delle funzioni di libreria
        // «__udivdi3()» e «__umoddi3()».
        //
        else if (format[f] == 'l')
          {
            if (format[f+1] == 'd' || format[f+1] == 'i')
              {
                // long int base 10.
                value_i = va_arg (ap, long int);
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 2;
              }
            else if (format[f+1] == 'u')
              {
                // Unsigned long int base 10.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
```



```
                f += 2;
              }
            else if (format[f+1] == 'o')
              {
                // Unsigned long int base 8.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'x')
              {
                // Unsigned long int base 16.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'X')
              {
                // Unsigned long int base 16.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'b')
              {
                // Unsigned long int base 2 (estensione).
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else // Specificatore errato;
              {
                f += 1;
              }
          }
        else if (format[f] == 'j')
          {
            if (format[f+1] == 'd' || format[f+1] == 'i')
              {
                // intmax_t base 10.
                value_i = va_arg (ap, intmax_t);
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 2;
              }
            else if (format[f+1] == 'u')
              {
                // uintmax_t base 10.
                value_ui = va_arg (ap, uintmax_t);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'o')
              {
                // uintmax_t base 8.
                value_ui = va_arg (ap, uintmax_t);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'x')
              {
                // uintmax_t base 16.
                value_ui = va_arg (ap, uintmax_t);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'X')
              {
                // uintmax_t base 16.
                value_ui = va_arg (ap, uintmax_t);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'b')
              {
                // uintmax_t base 2 (estensione).
                value_ui = va_arg (ap, uintmax_t);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else // Specificatore errato;
              {
                f += 1;
              }
          }
        else if (format[f] == 'z')
```

```
            {
              if (format[f+1] == 'd'
                  || format[f+1] == 'i'
                  || format[f+1] == 'i')
                {
                  // size_t base 10.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'o')
                {
                  // size_t base 8.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'x')
                {
                  // size_t base 16.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'X')
                {
                  // size_t base 16.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'b')
                {
                  // size_t base 2 (estensione).
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else // Specificatore errato;
                {
                  f += 1;
                }
            }
          else if (format[f] == 't')
            {
              if (format[f+1] == 'd' || format[f+1] == 'i')
                {
                  // ptrdiff_t base 10.
                  value_i = va_arg (ap, long int);
                  if (flag_plus)
                    {
                      s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                          alignment, filler, remain);
                    }
                  else
                    {
                      s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                         alignment, filler, remain);
                    }
                  f += 2;
                }
              else if (format[f+1] == 'u')
                {
                  // ptrdiff_t base 10, senza segno.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'o')
                {
                  // ptrdiff_t base 8, senza segno.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'x')
                {
                  // ptrdiff_t base 16, senza segno.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'X')
                {
                  // ptrdiff_t base 16, senza segno.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                      alignment, filler, remain);
                  f += 2;
                }
              else if (format[f+1] == 'b')
                {
                  // ptrdiff_t base 2, senza segno.
                  value_ui = va_arg (ap, unsigned long int);
                  s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
```



```
                                      alignment, filler, remain);
                  f += 2;
                }
              else // Specificatore errato;
                {
                  f += 1;
                }
            }
          if (format[f] == 'd' || format[f] == 'i')
            {
              // int base 10.
              value_i = va_arg (ap, int);
              if (flag_plus)
                {
                  s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                      alignment, filler, remain);
                }
              else
                {
                  s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                     alignment, filler, remain);
                }
              f += 1;
            }
          else if (format[f] == 'u')
            {
              // unsigned int base 10.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                  alignment, filler, remain);
              f += 1;
            }
          else if (format[f] == 'o')
            {
              // Unsigned int base 8.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                  alignment, filler, remain);
              f += 1;
            }
          else if (format[f] == 'x')
            {
              // unsigned int base 16.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                  alignment, filler, remain);
              f += 1;
            }
          else if (format[f] == 'X')
            {
              // unsigned int base 16.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                  alignment, filler, remain);
              f += 1;
            }
          else if (format[f] == 'b')
            {
              // unsigned int base 2 (estensione).
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                  alignment, filler, remain);
              f += 1;
            }
          //else if (format[f] == 'c')
          //  {
          //    // unsigned char.
          //    value_ui = va_arg (ap, unsigned int);
          //    s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
          //                        alignment, filler, remain);
          //    f += 1;
          //  }
          else if (format[f] == 'c')
            {
              // unsigned char.
              value_ui = va_arg (ap, unsigned int);
              string[s] = (char) value_ui;
              s += 1;
              f += 1;
            }
          else if (format[f] == 's')
            {
              // string.
              value_cp = va_arg (ap, char *);
              filler = ' ';

              s += strtostr_fill (value_cp, &string[s], alignment,
                                  filler, remain);
              f += 1;
            }
          else // Specificatore errato;
            {
              ;
            }
          //
          // Fine dello specificatore.
          //
          width_string[0]     = '\0';
          precision_string[0] = '\0';

          specifier           = 0;
          specifier_flags     = 0;
          specifier_width     = 0;
```

```
            specifier_precision = 0;
            specifier_type      = 0;

            flag_plus           = 0;
            flag_minus          = 0;
            flag_space          = 0;
            flag_alternate      = 0;
            flag_zero           = 0;
          }
      }
    string[s] = '\0';
    return s;
}
```

Listato u166.21. ‘./05/lib/snprintf.c’

```
#include <stdio.h>
int
snprintf (char *restrict string, size_t n, const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vsnprintf (string, n, format, ap);
}
```


Librerie specifiche generali

File «build.h» 1931
Libreria «io.h» 1931
Libreria «multiboot.h» 1932
File «os.h» 1933
Libreria «vga.h» 1935

build.h 1931 inb.s 1931 io.h 1931 mboot_info.c 1932 mboot_show.c 1932 multiboot.h 1932 os.h 1933 outb.s 1931 vga.h 1935 vga_clear.c 1935 vga_new_line.c 1935 vga_printf.c 1935 vga_putc.c 1935 vga_puts.c 1935 vga_set.c 1935 vga_vprintf.c 1935

Dopo le librerie standard vanno predisposte anche altre librerie specifiche per il proprio sistema. Quelle descritte nelle sezioni successive sono quelle di uso generale.

File «build.h»

Il file ‘05/include/kernel/build.h’ viene prodotto dallo script ‘05/makeit’, allo scopo di generare la macro-variabile ***BUILD_DATE*** contenente il momento esatto della compilazione. Durante gli esperimenti per la realizzazione del sistema è importante rendersi conto se ciò che si sta osservando corrisponde effettivamente al risultato dell’ultima compilazione oppure no. Il contenuto del file ha un aspetto simile a quello seguente:

```
#define BUILD_DATE "20070817191030"
```

Libreria «io.h»

La libreria rappresentata dal file di intestazione ‘io.h’ contiene la dichiarazione di funzioni necessarie alla comunicazione con le componenti hardware. In questo caso si utilizzano solo funzioni per riprodurre le istruzioni ‘**INB**’ e ‘**OUTB**’ del linguaggio assemblatore, ma potrebbe essere estesa anche con altre funzioni per istruzioni analoghe, per la comunicazione con dati di dimensione maggiore del byte.

Listato u167.2. ‘./05/include/kernel/io.h’

```
#ifndef _IO_H
#define _IO_H   1

void         outb (unsigned int port, unsigned int data);
unsigned int inb  (unsigned int port);

#endif
```

Naturalmente è necessario realizzare entrambe le funzioni. È il caso di ricordare che il valore restituito dalle funzioni scritte in linguaggio assemblatore è quello contenuto nel registro ***EAX***.

Listato u167.3. ‘./05/lib/io/inb.s’

```
.globl  inb
#
inb:
    enter $4, $0
    pusha
    .equ inb_port,  8           # Primo parametro.
    .equ inb_data, -4           # Variabile locale.
    mov  inb_port(%ebp), %edx   # Successivamente si usa
                                # solo DX.
    inb %dx, %al
    mov  %eax, inb_data(%ebp)   # Salva EAX nella variabile
                                # locale.
    popa
    mov  inb_data(%ebp), %eax   # Recupera EAX e termina.
    leave
    ret
```

Listato u167.4. ‘./05/lib/io/outb.s’

```
.globl  outb
#
outb:
    enter $0, $0
    pusha
    .equ outb_port,  8          # Primo parametro.
    .equ outb_data, 12          # Secondo parametro.
    mov  outb_port(%ebp), %edx  # Successivamente si usa
                                # solo DX.
    mov  outb_data(%ebp), %eax  # Successivamente si usa
                                # solo AL.
    outb %al, %dx
    popa
    leave
    ret
```

Libreria «multiboot.h»

La libreria rappresentata dal file di intestazione ‘multiboot.h’ contiene semplicemente una struttura per facilitare la lettura delle informazioni più importanti che offre un sistema di avvio aderente alle specifiche *multiboot*; inoltre dichiara due funzioni: una per la raccolta delle informazioni e l’altra per la loro visualizzazione.

Listato u167.5. ‘./05/include/kernel/multiboot.h’

```
#ifndef _MULTIBOOT_H
#define _MULTIBOOT_H    1

#include <inttypes.h>

typedef struct {
    uint32_t  flags;
    uint32_t  mem_lower;
    uint32_t  mem_upper;
    uint32_t  boot_device;
        char *cmdline;
} multiboot_t;

void mboot_info (multiboot_t *info);
void mboot_show (void);

#endif
```

La funzione ***mboot_info()*** deve raccogliere e salvare le informazioni *multiboot*, all’interno della variabile strutturata ***os.multiboot*** (la variabile ***os*** complessiva è descritta nel file ‘os.h’).

Listato u167.6. ‘./05/lib/multiboot/mboot_info.c’

```
#include <kernel/multiboot.h>
#include <string.h>
#include <stdio.h>
void
mboot_info (multiboot_t *info)
{
    os.multiboot.flags = info->flags;
    //
    if ((info->flags & 1) > 0)
      {
        os.multiboot.mem_lower = info->mem_lower;
        os.multiboot.mem_upper = info->mem_upper;
      }
    if ((info->flags & 2) > 0)
      {
        os.multiboot.boot_device = info->boot_device;
      }
    if ((info->flags & 4) > 0)
      {
        strncpy (os.multiboot.cmdline, info->cmdline, 1024);
      }
}
```

La funzione ***mboot_show()*** deve visualizzare direttamente le informazioni *multiboot*, salvate in precedenza, pertanto si avvale della funzione ***printf()*** che deve essere ancora descritta.

Listato u167.7. ‘./05/lib/multiboot/mboot_show.c’

```
#include <kernel/multiboot.h>
#include <stdio.h>
void
mboot_show (void)
{
    printf ("[%s] flags: %032b ", __func__,
            os.multiboot.flags);
    //
    if ((os.multiboot.flags & 1) > 0)
      {
        printf ("mlow: %04X mhigh: %08X",
                os.multiboot.mem_lower,
                os.multiboot.mem_upper);
      }
    printf ("\n");
    printf ("[%s] ", __func__);
    if ((os.multiboot.flags & 2) > 0)
      {
        printf ("bootdev: %08X ", os.multiboot.boot_device);
      }
    if ((os.multiboot.flags & 4) > 0)
      {
        printf ("cmdline: \"%s\"", os.multiboot.cmdline);
      }
    printf ("\n");
}
```

File «os.h»

Il file di intestazione ‘os.h’ serve esclusivamente per definire una struttura, con la quale si crea la variabile strutturata ***os***, accessibile a ogni parte del sistema. In questa superstruttura vengono annotate tutte le informazioni che devono essere condivise. Il senso delle varie componenti della variabile ***os*** si chiarisce successivamente; a ogni modo è importante osservare che nel sistema non vengono usate altre variabili pubbliche.

Listato u167.8. ‘./05/include/kernel/os.h’

```
#ifndef _OS_H
#define _OS_H   1

#include <stdint.h>
#include <kernel/multiboot.h>
#include <stdbool.h>
#include <time.h>

typedef struct {
    //
    // Multiboot.
    //
    struct {
        uint32_t  flags;
        uint32_t  mem_lower;
        uint32_t  mem_upper;
        uint32_t  boot_device;
        char      cmdline[1024];
    } multiboot;
    //
    // Stato dello schermo VGA.
    //
    struct {
        unsigned short *video;
        unsigned short  columns;
        unsigned short  rows;
        unsigned int    position;
        unsigned char   attribute;
    } vga;
    //
    // «os.mem_ph»    Mappa della memoria fisica.
    //
    struct {
        uintptr_t  total_s;     // «..._s» = start
        uintptr_t  total_e;     // «..._e» = end.
           size_t  total_l;     // «..._l» = limit.
        uintptr_t  k_text_s;    // «k_...» = kernel.
        uintptr_t  k_text_e;    //
        uintptr_t  k_rodata_s;  //
        uintptr_t  k_rodata_e;  //
        uintptr_t  k_data_s;    //
        uintptr_t  k_data_e;    //
        uintptr_t  k_bss_s;     //
        uintptr_t  k_bss_e;     //
        uintptr_t  available_s; //
        uintptr_t  available_e; //
    } mem_ph;
    //
```

```
    // «os.gtd»               Tabella GTD.
    //
    union {
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      accessed             : 1,
                      write_execute        : 1,
                      expansion_conforming : 1,
                      code_or_data         : 1,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      available            : 1,
                      reserved             : 1,
                      big                  : 1,
                      granularity          : 1,
                      base_c               : 8;
        } cd;
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      type                 : 4,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      reserved             : 3,
                      granularity          : 1,
                      base_c               : 8;
        } system;
    } gdt[3];
    //
    // «os.gtdr»      Registro GTDR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
    } __attribute__ ((packed)) gdtr;
    //
    // «os.idt»               Tabella IDT.
    //
    struct {
        uint32_t  offset_a : 16,
                  selector : 16;
        uint32_t  filler   : 8,
                  type     : 4,
                  system   : 1,
                  dpl      : 2,
                  present  : 1,
                  offset_b : 16;
    } idt[129];
    //
    // «os.idtr»      Registro IDTR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
    } __attribute__ ((packed)) idtr;
    //
    // PIT: programmable interval timer.
    //
    struct {
        clock_t freq;
        clock_t clocks;
    } timer;
    //
    // Stato della tastiera.
    //
    struct {
```



```
        bool shift;
        bool shift_lock;
        bool ctrl;
        bool alt;
        bool echo;
        char key;
        char map1[128];
        char map2[128];
    } kbd;
    //
} os_t;
//
// Struttura pubblica con tutte le informazioni sul sistema.
//
os_t os;

#endif
```

Libreria «vga.h»

La libreria che fa capo al file di intestazione '`vga.h`' è responsabile della visualizzazione del testo attraverso lo schermo.

Listato u167.9. '`./05/include/kernel/vga.h`'

```
#ifndef _VGA_H
#define _VGA_H 1

#include <restrict.h>
#include <kernel/io.h>
#include <kernel/os.h>
#include <stddef.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>

#define vga_char(c, attrib) \
    ((int16_t) c | ((((int16_t) attrib) << 8) & 0xFF00))

void vga_init     (void);
void vga_set      (unsigned short *video, int columns,
                   int rows, int x, int y, int position,
                   int attribute);
int  vga_clear    (void);
void vga_new_line (void);
void vga_putc     (int c);
void vga_puts     (char *string, size_t n);
int  vga_vprintf  (const char *restrict format,
                   va_list arg);
int  vga_printf   (const char *restrict format, ...);

#define clear()   (vga_clear ())
#define echo()    (os.kbd.echo = 1)
#define noecho()  (os.kbd.echo = 0)

#endif
```

Alcune macroistruzioni definite nel file '`vga.h`' si limitano a scrivere un valore all'interno di ***os.kbd.echo***, la quale, se attiva, rappresenta la richiesta di visualizzare sullo schermo il testo che viene digitato. La macroistruzione ***vga_char()*** assembla due valori in modo da ottenere un valore a 16 bit adatto alla visualizzazione sullo schermo di un carattere (l'unione dell'attributo di visualizzazione e del carattere stesso).

La funzione '`vga_init()`' va usata prima di fare qualunque cosa con lo schermo VGA, per attribuire dei valori iniziali corretti alla struttura ***os.vga***, la quale serve a tenere memoria della posizione corrente del cursore di scrittura e dell'attributo corrente da usare per i colori dei caratteri da scrivere.

Listato u167.10. '`./05/lib/vga/vga_init.c`'

```
#include <kernel/vga.h>
void
vga_init (void)
{
    os.vga.video     = (unsigned short *) 0xB8000;
    os.vga.columns   = 80;
    os.vga.rows      = 25;
    os.vga.position  = 0;
    os.vga.attribute = 0x07;
}
```

La funzione ‘`vga_set()`’ che appare nel listato successivo ha lo scopo di spostare e di tenere traccia della posizione corrente del cursore di scrittura, in base alle informazioni che gli vengono fornite, determinando il resto in modo predefinito. Va osservato che, quando si tratta di valori interi, per dire alla funzione ***vga_set()*** di utilizzare i dati predefiniti si trasmette un valore negativo.

Listato u167.11. ‘`./05/lib/vga/vga_set.c`’

```
#include <kernel/vga.h>
void
vga_set (unsigned short *video,
         int columns, int rows,
         int x, int y,
         int position,
         int attribute)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned int screen_size = os.vga.columns * os.vga.rows;
    char position_high;
    char position_low;
    //
    if (video != NULL)  os.vga.video   = video;
    if (columns >= 0)   os.vga.columns = columns;
    if (rows >= 0)      os.vga.rows    = rows;
    if (columns >= 0 || rows >= 0)
                        screen_size    = os.vga.columns * os.vga.rows;
    if (x >= 0)         current_x      = x;
    if (y >= 0)         current_y      = y;
    if (x >= 0 || y >= 0)
      {
        os.vga.position   = current_y * os.vga.columns + current_x;
        os.vga.position   = os.vga.position % screen_size;
      }
    if (position >= 0)
      {
        //
        // Ricalcola la posizione anche se è già stata determinata
        // con i parametri "x" and "y".
        //
        os.vga.position   = position % screen_size;
      }
    if (x >= 0 || y >= 0 || position >= 0)
      {
        //
        // Deve riposizionare il cursore.
        //
        position_high = (unsigned char) (os.vga.position >> 8);
        position_low  = (unsigned char) os.vga.position;
        //
        outb (0x3D4, 0x0E);
        outb (0x3D5, position_high);
        outb (0x3D4, 0x0F);
        outb (0x3D5, position_low);
      }
    if (attribute >= 0) os.vga.attribute = attribute;
}
```

Listato u167.12. ‘`./05/lib/vga/vga_clear.c`’

```
#include <kernel/vga.h>
int
vga_clear (void)
{
    unsigned short blank = vga_char (' ', os.vga.attribute);
    unsigned int   i;
    unsigned int   screen_size = os.vga.columns * os.vga.rows;

    for (i = 0; i < screen_size ; i++)
      {
        *(os.vga.video + i) = blank;
      }
    return 0;   // Per essere compatibile, in qualche modo, con «clear()».
}
```

Listato u167.13. ‘`./05/lib/vga/vga_new_line.c`’

```
#include <kernel/vga.h>
void
vga_new_line (void)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short blank       = vga_char (' ', os.vga.attribute);
    unsigned int   screen_size = os.vga.columns * os.vga.rows;
    int i;
    int j;

    current_x = 0;
    current_y++;

    if (current_y >= os.vga.rows)
      {
        //
        // Copia il testo in su di una riga.
        //
        for (i = 0, j = os.vga.columns; j < screen_size; i++, j++)
          {
            *(os.vga.video + i) = *(os.vga.video + j);
          }
        //
```



```
        // Ripulisce l'ultima riga di testo.
        //
        for (i = screen_size - os.vga.columns; i < screen_size; i++)
          {
            *(os.vga.video + i) = blank;
          }
        current_y--;
      }
    vga_set (NULL, -1, -1, current_x, current_y, -1, -1);
}
```

Listato u167.14. ‘`./05/lib/vga/vga_putc.c`’

```
#include <kernel/vga.h>
void
vga_putc (int c)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short int cell;

    if (c == '\n' || c == '\r')
      {
        vga_new_line ();
      }
    else
      {
        cell = vga_char (c, os.vga.attribute);

        *(os.vga.video + os.vga.position) = cell;

        if (current_x == os.vga.columns)
          {
            vga_new_line ();
          }
        else
          {
            vga_set (NULL, -1, -1, -1, -1, os.vga.position + 1, -1);
          }
      }
}
```

Listato u167.15. ‘`./05/lib/vga/vga_puts.c`’

```
#include <kernel/vga.h>
void
vga_puts (char *string, size_t n)
{
    size_t i;
    for (i = 0; i < n ; i++)
      {
        if (string[i] == 0) break;
        if (string[i] != 0) vga_putc (string[i]);
      }
    // Non aggiunge "\n"!
}
```

Listato u167.16. ‘`./05/lib/vga/vga_vprintf.c`’

```
#include <kernel/vga.h>
int
vga_vprintf (const char *restrict format, va_list arg)
{
    const size_t dim = 2000;  // Dimensione massima dello schermo: 25x80.
    char  string[dim];
    int   ret;
    string[0] = 0;
    ret = vsprintf(string, format, arg);
    vga_puts (string, dim);
    return ret;
}
```

Listato u167.17. ‘`./05/lib/vga/vga_printf.c`’

```
#include <kernel/vga.h>
int
vga_printf  (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vga_vprintf (format, ap);
}
```

File «kernel.h» 1939
Altri file mancanti 1942
Compilazione e prova di funzionamento 1942

`kernel.h` 1939 `kernel_boot.s` 1939 `kernel_memory.c` 1939 `_Exit.s` 1942

Avviando il sistema con GRUB 1 o con un altro programma conforme alle specifiche *multiboot*, il kernel dovrebbe trovarsi già in un contesto funzionante in modalità protetta, utilizzando tutta la memoria in modo lineare (ovvero senza suddivisione in segmenti). Pertanto, per visualizzare qualcosa sullo schermo non è indispensabile il passare subito alla preparazione della tabella GDT, cosa che consente di verificare se i file già preparati sono corretti.

In queste sezioni vengono descritti altri file del sistema in fase di sviluppo, ma in particolare '`kernel_main.c`' non è ancora nella sua impostazione definitiva, per consentire una verifica provvisoria del lavoro.

File «kernel.h»

Il file di intestazione '`kernel.h`' viene usato soprattutto per definire le funzioni principali del kernel, ma si possono notare, in coda, delle funzioni che in realtà non esistono, corrispondenti a simboli generati attraverso il «collegatore» (il *linker*). Queste funzioni fantasma servono solo per consentire l'individuazione degli indirizzi rispettivi, così da sapere come è disposto in memoria il kernel.

Listato u168.1. '`./05/include/kernel/kernel.h`'

```
#ifndef _KERNEL_H
#define _KERNEL_H       1

#include <restrict.h>
#include <kernel/multiboot.h>
#include <kernel/os.h>
//
// Funzioni principali da cui inizia l'esecuzione del kernel.
//
void kernel_boot        (void);
void kernel_main        (unsigned long magic, multiboot_t *info);
void kernel_memory      (multiboot_t *info);
void kernel_memory_show (void);
//
// Simboli di riferimento inseriti dallo script di LD (linker script).
// Vengono dichiarate qui come funzioni, solo per comodità, ma servono
// solo per individuare le posizioni utilizzate dal kernel nella memoria
// fisica, così da poter costruire poi una tabella GDT decente.
//
void k_mem_total_s  (void);
void k_mem_text_s   (void);
void k_mem_text_e   (void);
void k_mem_rodata_s (void);
void k_mem_rodata_e (void);
void k_mem_data_s   (void);
void k_mem_data_e   (void);
void k_mem_bss_s    (void);
void k_mem_bss_e    (void);
void k_mem_total_e  (void);

#endif
```

La funzione ***kernel_boot()*** è quella responsabile dell'avvio ed è scritta necessariamente in linguaggio assemblatore. Si trova contenuta nel file '`kernel_boot.s`', assieme alla dichiarazione dell'impronta di riconoscimento *multiboot* e alla collocazione dello spazio usato per la pila dei dati (l'unica pila che questo piccolo sistema utilizzi). È attraverso la configurazione del collegatore, nel file '`linker.ld`', che viene specificato di partire con la funzione ***kernel_boot()***.

Listato u168.2. '`./05/kernel/kernel_boot.s`'

```
.extern kernel_main
#
.globl  kernel_boot
#
# Dimensione della pila interna al kernel. Qui vengono previsti
# 32768 byte (0x8000 byte).
#
.equ STACK_SIZE,  0x8000
```

```
#
# Si inizia subito con il codice che si mescola con i dati;
# pertanto si deve saltare alla procedura che deve predisporre
# la pila e avviare il kernel scritto in C.
#
kernel_boot:
    jmp start
#
# Per collocare correttamente i dati che si trovano dopo l'istruzione
# di salto, si fa in modo di riempire lo spazio mancante al
# completamento di un blocco di 4 byte.
#
.align 4
#
# Intestazione «multiboot» che deve apparire poco dopo l'inizio
# del file-immagine.
#
multiboot_header:
    .int 0x1BADB002                     # magic
    .int 0x00000003                     # flags
    .int -(0x1BADB002 + 0x00000003)     # checksum
#
# Inizia il codice di avvio.
#
start:
    #
    # Regola ESP alla base della pila.
    #
    movl $(stack_max + STACK_SIZE), %esp
    #
    # Azzera gli indicatori contenuti in EFLAGS, ma per questo deve
    # usare la pila appena sistemata.
    #
    pushl $0
    popf
    #
    # Chiama la funzione principale scritta in C, passandogli le
    # informazioni ottenute dal sistema di avvio.
    #
    # void kernel_main (unsigned int magic, void *multiboot_info)
    #
    pushl %ebx          # Puntatore alla struttura contenente le
                        # informazioni passate dal sistema di avvio.
    pushl %eax          # Codice di riconoscimento del sistema di avvio.
    #
    call kernel_main    # Chiama la funzione kernel().
#
# Procedura di arresto.
#
halt:
    hlt                 # Se il kernel termina, ferma il microprocessore.
    jmp halt            # Se il microprocessore viene sbloccato, si
                        # ripete il comando HLT.
#
# Alla fine viene collocato lo spazio per la pila dei dati,
# senza inizializzarlo. Per scrupolo si allinea ai 4 byte (32 bit).
#
.align 4
.comm stack_max, STACK_SIZE
```

La funzione ***kernel_main()*** (avviata da ***kernel_boot()***) che viene mostrata nel listato successivo, non è ancora nella sua forma definitiva: per il momento si limita alla visualizzazione delle informazioni ***multiboot*** e allo stato della memoria utilizzata.

Listato u168.3. Prima versione del file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
void
kernel_main (unsigned long magic, multiboot_t *info)
{
    //
    // Inizializza i dati relativi alla gestione dello
    // schermo VGA, quindi ripulisce lo schermo.
    //
    vga_init ();
    clear ();
    //
    // Data e orario di compilazione.
    //
    printf ("05 %s\n", BUILD_DATE);
    //
    // Cerca le informazioni «multiboot».
    //
    if (magic == 0x2BADB002)
      {
        //
        // Salva e mostra le informazioni multiboot.
        //
```



```
        mboot_info (info);
        mboot_show ();
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Omissis.
        //
      }
    else
      {
        printf ("[%s] no \"multiboot\" header!\n",
                __func__);
      }
    //
    printf ("[%s] system halted\n", __func__);
    _Exit (0);
}
```

I listati successivi, relativi alle funzioni ***kernel_memory()*** e ***kernel_memory_show()***, sono nel loro stato definitivo.

Listato u168.4. '`./05/kernel/kernel_memory.c`'

```
#include <kernel/kernel.h>
#include <stdio.h>
void
kernel_memory (multiboot_t *info)
{
    //
    // Imposta valori conosciuti o predefiniti.
    //
    os.mem_ph.total_s     = (uint32_t) &k_mem_total_s;
    os.mem_ph.total_e     = (uint32_t) &k_mem_total_e;
    os.mem_ph.available_s = (uint32_t) &k_mem_total_e;
    os.mem_ph.available_e
     = (uint32_t) &k_mem_total_e+0x0FFFFF; // 1 Mibyte.
    //
    os.mem_ph.k_text_s   = (uint32_t) &k_mem_text_s;
    os.mem_ph.k_text_e   = (uint32_t) &k_mem_text_e;
    os.mem_ph.k_rodata_s = (uint32_t) &k_mem_rodata_s;
    os.mem_ph.k_rodata_e = (uint32_t) &k_mem_rodata_e;
    os.mem_ph.k_data_s   = (uint32_t) &k_mem_data_s;
    os.mem_ph.k_data_e   = (uint32_t) &k_mem_data_e;
    os.mem_ph.k_bss_s    = (uint32_t) &k_mem_bss_s;
    os.mem_ph.k_bss_e    = (uint32_t) &k_mem_bss_e;
    //
    if ((info->flags & 1) > 0)
      {
        os.mem_ph.available_e = 1024 * info->mem_upper;
      }
    //
    os.mem_ph.total_l = os.mem_ph.available_e / 0x1000;
    //
    kernel_memory_show ();
}
```

Listato u168.5. '`./05/kernel/kernel_memory_show.c`'

```
#include <kernel/kernel.h>
#include <stdio.h>
void
kernel_memory_show (void)
{
    //
    printf ("[%s] kernel %08" PRIX32 "..%08" PRIX32
                "  avail. %08" PRIX32 "..%08" PRIX32 "\n",
                        __func__,
                        os.mem_ph.total_s,
                        os.mem_ph.total_e,
                        os.mem_ph.available_s,
                        os.mem_ph.available_e);
    //
    printf ("[%s] text   %08" PRIX32 "..%08" PRIX32
                "  rodata %08" PRIX32 "..%08" PRIX32 "\n",
                        __func__,
                        os.mem_ph.k_text_s,
                        os.mem_ph.k_text_e,
                        os.mem_ph.k_rodata_s,
                        os.mem_ph.k_rodata_e);
    //
```

```
    printf ("[%s] data   %08" PRIX32 "..%08" PRIX32
                "  bss    %08" PRIX32 "..%08" PRIX32 "\n",
                        __func__,
                        os.mem_ph.k_data_s,
                        os.mem_ph.k_data_e,
                        os.mem_ph.k_bss_s,
                        os.mem_ph.k_bss_e);
    //
    printf ("[%s] limit  %08" PRIX32 "\n",
                        __func__,
                        os.mem_ph.total_l);
}
```

Altri file mancanti

Nella descrizione della libreria che fa capo al file di intestazione '`stdlib.h`', è stata omessa la funzione ***_Exit()*** che ora è indispensabile precisare, essendo usata dalla funzione ***kernel_main()***. In pratica si esegue semplicemente un ciclo senza fine, cercando però di sospendere il funzionamento del microprocessore, fino a quando si verifica un'interruzione.

Listato u168.6. '`./05/lib/_Exit.s`'

```
.globl _Exit
#
_Exit:
    enter $0, $0
    .equ status,  8         # Primo argomento.
    mov  status(%ebp), %eax # Copia il valore da restituire
                            # in EAX, anche se poi non se ne
                            # fa nulla.
    leave
    #
halt:
    hlt         # Ferma il microprocessore.
    jmp halt    # Se il microprocessore viene sbloccato, si
                # ripete il comando HLT.
```

Compilazione e prova di funzionamento

Prima di procedere alla compilazione con lo script '**`compile`**' (o direttamente con '**`makeit`**'), occorre verificare che la variabile di ambiente '**`TAB`**' sia dichiarata correttamente nello script '**`makeit`**', in modo da contenere esattamente un carattere di tabulazione orizzontale (diversamente i file-make non verrebbero creati nel modo giusto). Inoltre occorre avere preparato il file-immagine del dischetto e averlo innestato nella directory '`/mnt/fd0/`' (diversamente occorre modificare sempre lo script '**`makeit`**'). Quando tutto sembra pronto, basta avviare lo script '**`bochs`**' (da una finestra di terminale, durante una sessione grafica di lavoro con X) per far partire il sistema giocattolo in prova. Se tutto va bene, viene visualizzato il testo seguente e poi tutto si ferma; se invece si presenta un errore, il simulatore Bochs riavvia e si riparte con GRUB 1.

```
05 20070818140007
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010BAFC  avail. 0010BAFC..01EF0000
[kernel_memory_show] text   00100000..00102FEC  rodata 00102FEC..00103144
[kernel_memory_show] data   00103144..00103144  bss    00103160..0010BAFC
[kernel_memory_show] limit  00001EF0
[kernel_main] system halted
```

Dall'esempio mostrato si può determinare quanto segue: la memoria bassa arriva fino a $27F_{16}$ Kibyte (639 Kibyte); la memoria alta arriva fino a $7BC0_{16}$ Kibyte (31680 Kibyte); il kernel utilizza la memoria da 100000_{16} byte (1024 Kibyte) a $10BAFC_{16}$ byte (1070 Kibyte circa); pertanto la parte rimanente è tutta memoria libera.

Con questi dati, nel prossimo gruppo di sezioni viene preparata una tabella GDT minima, con la quale si definisce solo la memoria esistente effettivamente.

Tabella GDT

Struttura 1943
Libreria «gdt.h» 1943
Modifiche da apportare a «kernel_main.c» 1946

`gdt.c` 1943 `gdt.h` 1943 `gdt_desc_seg.c` 1943 `gdt_load.s` 1943 `gdt_print.c` 1943

La preparazione di una tabella GDT è indispensabile per poter accedere alla memoria in modalità protetta. Nel sistema in corso di realizzazione si intende usare la memoria in modo lineare, senza segmentazioni e senza pagine, pertanto si compila la tabella GDT con il minimo indispensabile, avendo cura di indicare in modo preciso la memoria esistente effettivamente.

Struttura

Nel file di intestazione '`os.h`' è già stata predisposta la struttura che facilita la compilazione e l'interpretazione dei descrittori della tabella GDT. In particolare viene usata un'unione, con due suddivisioni alternative: una per i descrittori di segmento codice o dati e l'altra per i descrittori di sistema. Per il lavoro in corso di realizzazione, i descrittori di sistema non vengono utilizzati, pertanto è sufficiente concentrarsi sulla struttura '**`os.gdt[`*`n`*`].cd`**':

```
…
    union {
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      accessed             : 1,
                      write_execute        : 1,
                      expansion_conforming : 1,
                      code_or_data         : 1,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      available            : 1,
                      reserved             : 1,
                      big                  : 1,
                      granularity          : 1,
                      base_c               : 8;
        } cd;
        struct {
            …
            …
        } system;
    } gdt[3];
…
```

Libreria «gdt.h»

Il file di intestazione '`gdt.h`' contiene la dichiarazione delle funzioni che riguardano la gestione della tabella GDT.

Listato u169.2. '`./05/include/kernel/gdt.h`'

```
#ifndef _GDT_H
#define _GDT_H  1

#include <inttypes.h>
#include <stdbool.h>
#include <kernel/os.h>

void gdt_desc_seg (int           descriptor,
                   uint32_t      base,
                   uint32_t      limit,
                   bool          present,
                   bool          granularity,
                   bool          code,
                   bool          write_execute,
                   bool          expand_down_non_conforming,
                   unsigned char dpl);
void gdt_print      (void *gdtr);
void gdt_load       (void *gdtr);
```

```
void gdt            (void);

#endif
```

La funzione ***gdt_desc_seg()*** serve a facilitare la compilazione di un descrittore della tabella; la funzione ***gdt_print()*** consente di visualizzare il contenuto della tabella, partendo dal contenuto del registro ***GDTR***, indipendentemente da altre informazioni; la funzione ***gdt_load()*** fa in modo che il microprocessore utilizzi il contenuto della tabella GDT; la funzione ***gdt()***, avvalendosi delle altre funzioni già citate, crea la tabella minima richiesta, ne mostra il contenuto e la attiva.

Listato u169.3. ‘`./05/lib/gdt/gdt_desc_seg.c`’

```
#include <kernel/gdt.h>
#include <stdio.h>
void
gdt_desc_seg (int           desc,
              uint32_t      base,
              uint32_t      limit,
              bool          present,
              bool          granularity,
              bool          code,
              bool          write_execute,
              bool          expand_down_non_conforming,
              unsigned char dpl)
{
    //
    // Verifica di non eccedere la dimensione dell'array.
    //
    int max = ((sizeof (os.gdt)) / 8) - 1;
    if (desc > max)
      {
        printf ("[%s] ERROR: selected descriptor %i when max is %i!\n",
                __func__, desc, max);
        return;
      }
    //
    // Limite.
    //
    os.gdt[desc].cd.limit_a = (limit & 0x0000FFFF);
    os.gdt[desc].cd.limit_b = limit / 0x10000;
    //
    // Indirizzo base.
    //
    os.gdt[desc].cd.base_a = (base & 0x0000FFFF);
    os.gdt[desc].cd.base_b = ((base / 0x10000) & 0x000000FF);
    os.gdt[desc].cd.base_c = (base / 0x1000000);
    //
    // Attributi.
    //
    os.gdt[desc].cd.accessed             = 0;
    os.gdt[desc].cd.write_execute        = write_execute;
    os.gdt[desc].cd.expansion_conforming = expand_down_non_conforming;
    os.gdt[desc].cd.code_or_data         = code;
    os.gdt[desc].cd.code_data_or_system  = 1;
    os.gdt[desc].cd.dpl                  = dpl;
    os.gdt[desc].cd.present              = present;
    os.gdt[desc].cd.available            = 0;
    os.gdt[desc].cd.reserved             = 0;
    os.gdt[desc].cd.big                  = 1;
    os.gdt[desc].cd.granularity          = granularity;
}
```

Listato u169.4. ‘`./05/lib/gdt/gdt_print.c`’

```
#include <kernel/gdt.h>
#include <stdio.h>
//
// Mostra il contenuto di una tabella GDT, a partire dal puntatore al
// registro GDTR in memoria. Pertanto non si avvale, volutamente, della
// struttura già predisposta con il linguaggio C, mentre «gdtr_t» viene
// creato qui solo provvisoriamente, per uso interno. Ciò serve ad
// assicurare che questa funzione compia il proprio lavoro in modo
// indipendente, garantendo la visualizzazione di dati reali.
//
typedef struct {
    uint16_t  limit;
    uint32_t  base;
} __attribute__ ((packed)) local_gdtr_t;
//
void
gdt_print (void *gdtr)
{
    local_gdtr_t *g = gdtr;
        uint32_t *p = (uint32_t *) g->base;

    int max = (g->limit + 1) / (sizeof (uint32_t));
    int i;
```



```
    printf ("[%s] base: 0x%08" PRIX32 " limit: 0x%04" PRIX32 "\n",
            __func__, g->base, g->limit);

    for (i = 0; i < max; i+=2)
      {
        printf ("[%s] %" PRIx32 " %032" PRIb32 " %032" PRIb32 "\n",
                __func__, i/2, p[i], p[i+1]);
      }
}
```

Listato u169.5. ‘`./05/lib/gdt/gdt_load.s`’

```
.globl  gdt_load
#
gdt_load:
    enter $0, $0
    .equ gdtr_pointer,  8          # Primo argomento.
    mov  gdtr_pointer(%ebp), %eax  # Copia il puntatore
                                   # in EAX.
    leave
    #
    lgdt (%eax)   # Carica il registro GDTR dall'indirizzo
                  # in EAX.
    #
    # 2 dati per il kernel, DPL 0, comprendente tutta la
    #   memoria disponibile: selettore 0x10+0.
    #
    mov  $0x10, %ax
    mov  %ax,   %ds
    mov  %ax,   %es
    mov  %ax,   %fs
    mov  %ax,   %gs
    mov  %ax,   %ss
    #
    # 1 codice per il kernel, DPL 0, comprendente tutta
    #   la memoria disponibile: selettore 0x08+0.
    #
    jmp  $0x08, $flush
flush:
    ret
```

Listato u169.6. ‘`./05/lib/gdt/gdt.c`’

```
#include <kernel/gdt.h>
void
gdt (void)
{
    //
    // Imposta i dati necessari al registro GDTR.
    //
    os.gdtr.limit = (sizeof (os.gdt) - 1);
    os.gdtr.base  = (uint32_t) &os.gdt[0];
    //
    // Azzera le voci previste dell'array «os.gdt[]».
    // La prima di queste voci (0) rimane azzerata e non
    // deve essere utilizzata.
    //
    int i;
    for (i = 0; i < ((sizeof (os.gdt)) / 8); i++)
      {
        gdt_desc_seg (i, 0, 0, 0, 0, 0, 0, 0, 0);
      }
    //
    // 1 codice per il kernel, DPL 0, comprendente tutta la
    //   memoria disponibile: selettore 0x08+0.
    //
    gdt_desc_seg (1, 0,
                  os.mem_ph.total_l, 1, 1, 1, 1, 0, 0);
    //
    // 2 dati per il kernel, DPL 0, comprendente tutta la
    //   memoria disponibile: selettore 0x10+0.
    //
    gdt_desc_seg (2, 0,
                  os.mem_ph.total_l, 1, 1, 0, 1, 0, 0);
    //
    // Mostra la tabella GDT e poi la carica.
    //
    gdt_print (&os.gdtr);
    gdt_load  (&os.gdtr);
}
```

Modifiche da apportare a «kernel_main.c»

Nel file ‘`kernel_main.c`’ va aggiunta l’incorporazione del file ‘`gdt.h`’ e la chiamata alla funzione ***gdt()***:

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
…
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
…
```

Una volta ricompilato il lavoro e avviato con Bochs, si deve ottenere una schermata simile a quella seguente:

```
05 20070819115151
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010BF7C  avail. 0010BF7C..01EF0000
[kernel_memory_show] text   00100000..00103418  rodata 00103418..001035FC
[kernel_memory_show] data   001035FC..001035FC  bss    00103600..0010BF7C
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010B648 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[kernel_main] system halted
```


Gestione della memoria

Gestione della memoria attraverso una lista 1947
Libreria «mm.h» 1948
Funzioni per l’allocazione della memoria 1950
Verifica del funzionamento 1953

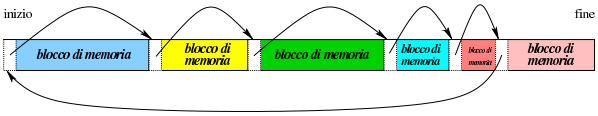
`free.c` 1950 `malloc.c` 1950 `mm.h` 1948 `mm_init.c` 1948 `mm_list.c` 1948 `realloc.c` 1950

Nel sistema in corso di realizzazione non si intende gestire la memoria in modo sofisticato; in particolare non si vogliono usare né segmenti, né pagine. In pratica, lo spazio che rimane dopo l’intervallo usato dal kernel viene gestito con una lista e sulla base di questa impostazione vengono realizzate le funzioni ‘…**`alloc()`**’ e ***free()***.

Gestione della memoria attraverso una lista

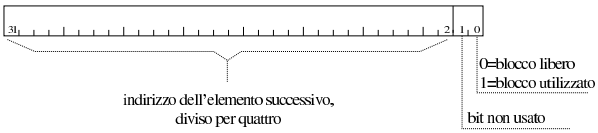
Si intende gestire l’allocazione di memoria attraverso una lista in cui l’inizio di un blocco di memoria contenga il riferimento al blocco successivo e l’indicazione se il proprio sia un blocco libero o utilizzato; pertanto, come si vede nella figura, l’ultimo blocco punta al primo.

Figura u170.1. Lista di blocchi di memoria.



L’intestazione dei blocchi di memoria, con la quale si fa riferimento al blocco successivo e si annota se il blocco (proprio) è impegnato o meno, utilizza solo 32 bit, partendo dal presupposto che i blocchi di memoria debbano essere multipli di tale valore. A tale proposito si osservi che se i blocchi di memoria sono da 4 byte, gli indirizzi sono sempre multipli di quattro, ovvero di 100_2. Pertanto, i due bit meno significativi possono essere utilizzati per altri fini.

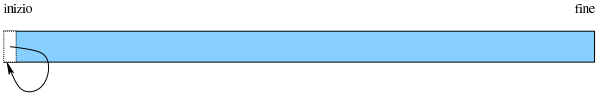
Figura u170.2. Struttura dell’intestazione dei blocchi di memoria.



Sulla base del principio affermato e di quanto si vede nella figura, l’indirizzo effettivo del blocco successivo si determina moltiplicando per quattro il valore annotato tra il bit 2 e il bit 31 dell’intestazione. Si osservi che l’indirizzo in questione è quello dell’inizio dell’intestazione del blocco successivo, pertanto il blocco di memoria successivo inizia effettivamente dopo altri quattro byte.

Quando la memoria viene inizializzata si crea un blocco solo, la cui intestazione punta a se stessa, come si vede nella figura successiva. Da questo si comprende anche che il blocco che punta a se stesso è lungo fino alla fine dello spazio di memoria disponibile complessivamente; inoltre si intende che con questo meccanismo, molto semplice, la memoria possa essere gestita solo se è presente in modo continuo. Questa semplificazione è stata fatta volutamente per non complicare inutilmente il codice; d’altra parte si osserva che così la «memoria bassa» (quella dei primi 640 Kibyte) non venga usata affatto.

Figura u170.3. La lista al momento iniziale.



Libreria «mm.h»

Il file di intestazione '`mm.h`' descrive la struttura usata per interpretare i primi 32 bit dei blocchi di memoria (per distinguere l'indirizzo successivo dall'indicazione dello stato del blocco attuale) e dichiara due funzioni per inizializzare la memoria e per leggerne la mappa.

Listato u170.4. '`./05/include/kernel/mm.h`'

```
#ifndef _MM_H
#define _MM_H 1

#include <restrict.h>
#include <stdint.h>
#include <inttypes.h>
#include <stddef.h>
#include <stdarg.h>
#include <kernel/os.h>

//
// La dimensione di «uintptr_t» condiziona la struttura
// «mm_head_t» e la dimensione delle unità minime di memoria
// allocata. «uintptr_t» è da 32 bit, così l'immagine del
// kernel è allineata a blocchi da 32 bit e così deve essere
// anche per gli altri blocchi di memoria. Essendo i blocchi
// di memoria multipli di 32 bit, gli indirizzi sono sempre
// multipli di 4 (4 byte); pertanto, servono solo 30 bit
// per rappresentare l'indirizzo, che poi viene ottenuto
// moltiplicandolo per quattro.
// Di conseguenza, il bit meno significativo viene usato
// per annotare se il blocco di memoria è libero e il bit
// successivo non viene usato. Questo meccanismo potrebbe
// essere usato anche con un indirizzamento a 16 bit, dove
// servirebbero 15 bit per indirizzi multipli di due byte.
//
typedef struct {
    uintptr_t allocated : 1,
              filler    : 1,
              next      : 30;
} mm_head_t;

void mm_init (void);
void mm_list (void);

#endif
```

La funzione ***mm_init()*** inizializza la memoria, creando un blocco libero che la descrive completamente. Per sapere dove inizia e dove finisce la memoria disponibile, si avvale delle informazioni contenute nella variabile strutturata ***os.mem_ph***, le quali sono state inserite precedentemente dalla funzione ***kernel_memory()*** (listato u168.4)

Listato u170.5. '`./05/lib/mm/mm_init.c`'

```
#include <kernel/mm.h>
#include <stdio.h>
void
mm_init (void)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head;
       size_t  available = os.mem_ph.available_e - os.mem_ph.available_s;
    //
    // La memoria disponibile deve essere di almeno 8 byte!
    //
    if (available < ((sizeof (mm_head_t)) * 2))
      {
        //
        // Il sistema viene fermato!
        //
        printf ("[%s] ERROR: not enough memory: %zu byte!\n",
                __func__, available);
        _Exit (0);
      }
    //
    // Predispone il nodo principale della lista.
    //
    head = (mm_head_t *) start;
    //
    // Inizializza il primo blocco, libero, che punta a se stesso,
    // essendo l'unico.
    //
    head->allocated = 0;
    head->next      = (start / (sizeof (mm_head_t)));
    //
    // Mostra come è andata.
    //
    printf ("[%s] available memory: %zu byte\n",
                __func__, available - (sizeof (mm_head_t)));
    //
    return;
}
```

La funzione ***mm_list()*** mostra la mappa della memoria gestita attraverso le funzioni '…**`alloc()`**'. Gli indirizzi che vengono forniti sono quelli di inizio dei blocchi, escludendo lo spazio utilizzato dalle intestazioni (pertanto, se l'intestazione inizia all'indirizzo ***n***, viene mostrato l'indirizzo ***n***+4).

Listato u170.6. '`./05/lib/mm/mm_list.c`'

```
#include <kernel/mm.h>
#include <stdio.h>
void mm_list (void)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head  = (void *) start;
       size_t  actual_size;
    uintptr_t  current;
    uintptr_t  next;
    uintptr_t  up_to;
          int  counter;
    //
    // Scandisce la lista di blocchi di memoria.
    //
    counter = 2;
    while (counter)
      {
        //
        // Annota la posizione attuale e quella successiva.
        //
        current = (uintptr_t) head;
           next = head->next * (sizeof (mm_head_t));
        if (next == start)
          {
            up_to = os.mem_ph.available_e;
          }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento, decrementa il
        // contatore di una unità. Se è già a zero, esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0) break;
          }
        //
        // Determina la dimensione del blocco attuale.
        //
        if (current == start && next == start)
          {
            //
            // Si tratta del primo e unico elemento della lista.
            //
            actual_size = os.mem_ph.available_e - (sizeof (mm_head_t));
          }
        else
          {
            actual_size = up_to - current - (sizeof (mm_head_t));
          }
        //
        // Si mostra lo stato del blocco di memoria.
        //
        if (head->allocated)
          {
            printf ("[%s] used %08X..%08X size %08zX\n",
                    __func__,
                    current + (sizeof (mm_head_t)), up_to, actual_size);
          }
        else
          {
            printf ("[%s] free %08X..%08X size %08zX\n",
                    __func__,
                    current + (sizeof (mm_head_t)), up_to, actual_size);
          }
        //
        // Si passa alla posizione successiva.
        //
        head = (void *) next;
      }
}
```

«

La funzione ***malloc()*** esegue una scansione della mappa della memoria, alla ricerca del primo blocco di dimensione sufficiente a soddisfare la richiesta ricevuta (*first fit*). Una volta trovato, se il blocco libero è abbastanza grande, lo divide, in modo da utilizzare solo lo spazio richiesto. Gli spazi allocati sono sempre multipli della dimensione di '**`mm_head_t`**', pertanto, se necessario, si alloca uno spazio leggermente più grande del richiesto.

Listato u170.7. '`./05/lib/malloc.c`'

```
#include <stdlib.h>
#include <kernel/mm.h>
void
*malloc (size_t size)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head  = (void *) start;
       size_t  actual_size;
    uintptr_t  current;
    uintptr_t  next;
    uintptr_t  new;
    uintptr_t  up_to;
          int  counter;
    //
    // Arrotonda in eccesso il valore di «size», in modo che sia un
    // multiplo della dimensione di «mm_head_t». Altrimenti, la
    // collocazione dei blocchi successivi può avvenire in modo
    // non allineato.
    //
    size = (size + (sizeof (mm_head_t)) -1);
    size = size / (sizeof (mm_head_t));
    size = size * (sizeof (mm_head_t));
    //
    // Cerca un blocco libero di dimensione sufficiente.
    //
    counter = 2;
    while (counter)
      {
        //
        // Annota la posizione attuale e quella successiva.
        //
        current = (uintptr_t) head;
           next = head->next * (sizeof (mm_head_t));
        //
        if (next == start)
          {
            up_to = os.mem_ph.available_e;
          }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento, decrementa il
        // contatore di una unità. Se è già a zero, esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0) break;
          }
        //
        // Controlla se si tratta di un blocco libero.
        //

        if (! head->allocated)
          {
            //
            // Il blocco è libero: si deve determinarne la dimensione.
            //
            if (current == start && next == start)
              {
                //
                // Si tratta del primo e unico elemento della lista.
                //
                actual_size = os.mem_ph.available_e - (sizeof (mm_head_t));
              }
            else
              {
                actual_size = up_to - current - (sizeof (mm_head_t));
              }
            //
            // Si verifica che sia capiente.
            //
            if (actual_size >= size + ((sizeof (mm_head_t)) * 2))
              {
                //
                // C'è spazio per dividere il blocco.
                //
                new = current + size + (sizeof (mm_head_t));
                //
                // Aggiorna l'intestazione attuale.
                //
                head->allocated = 1;
                head->next = new / (sizeof (mm_head_t));
                //
                // Predispone l'intestazione successiva.
                //
                head = (void *) new;
                head->allocated = 0;
                head->next = next / (sizeof (mm_head_t));
                //
                // Restituisce l'indirizzo iniziale dello spazio libero,
                // successivo all'intestazione.
                //
                return (void *) (current + (sizeof (mm_head_t)));
              }
            else if (actual_size >= size)
              {
                //
                // Il blocco va usato per intero.
                //
                head->allocated = 1;
                //
                // Restituisce l'indirizzo iniziale dello spazio libero,
                // successivo all'intestazione.
                //
                return (void *) (current + (sizeof (mm_head_t)));
              }
          }
        //
        // Il blocco è allocato, oppure è di dimensione insufficiente;
        // pertanto occorre passare alla posizione successiva.
        //
        head = (void *) next;
      }
    //
    // Essendo terminato il ciclo precedente, vuol dire
    // che non ci sono spazi disponibili.
    //
    return NULL;
}
```


La funzione ***free()*** libera il blocco di memoria indicato e poi scandisce tutti i blocchi esistenti alla ricerca di quelli liberi che sono adiacenti, per fonderli assieme. Va osservato che la funzione non verifica se il blocco da liberare esiste effettivamente e per evitare errori occorrerebbe una scansione preventiva dei blocchi, a partire dall'inizio.

Listato u170.8. '`./05/lib/free.c`'

```
#include <stdlib.h>
#include <kernel/mm.h>
#include <stdio.h>
void
free (void *ptr)
{
    mm_head_t *start        = (mm_head_t *) os.mem_ph.available_s;
    mm_head_t *head_current = ((mm_head_t *) ptr) - 1;
    mm_head_t *head_next;
    //
    // Verifica il blocco attuale e, se è possibile, lo libera.
    //
    if (head_current->allocated == 1)
      {
        head_current->allocated = 0;
      }
    else
      {
        printf ("[%s] ERROR: cannot free %08X!\n",
                __func__, (uintptr_t) head_current + (sizeof (mm_head_t)));
      }
    //
    // Scandisce i blocchi liberi, cercando quelli adiacenti per
    // allungarli. Se il blocco successivo è il primo, termina,
    // perché non può avvenire alcuna fusione con quello precedente.
    //
    head_current = start;
    while (true)
      {
        //
        // Individua il blocco successivo.
        //
        head_next = (mm_head_t *) (head_current->next * (sizeof (mm_head_t)));
        //
        // Controlla se è il primo.
        //
        if (head_next == start)
          {
            break;
          }
        //
        //
        //
        if (head_current->allocated == 0)
          {
            //
            // Controlla se si può espandere.
```

```
            //
            if (head_next->allocated == 0)
              {
                head_current->next = head_next->next;
              }
            else
              {
                head_current = head_next;
              }
          }
        else
          {
            head_current = (mm_head_t *)
                           (head_current->next * (sizeof (mm_head_t)));
          }
      }
}
```

La funzione ***realloc()***, nel caso sia richiesto un blocco più grande del precedente, si avvale di ***malloc()***, ***memcpy()*** e ***free()***.

Listato u170.9. '`./05/lib/realloc.c`'

```
#include <stdlib.h>
#include <string.h>
#include <kernel/mm.h>
#include <stdio.h>
void
*realloc (void *ptr, size_t size)
{
    uintptr_t  start     = os.mem_ph.available_s;
       size_t  actual_size;
    mm_head_t *head      = ((mm_head_t *) ptr) - 1;
    mm_head_t *head_new;
         void *ptr_new;
    //
    // Verifica che il puntatore riguardi effettivamente
    // un'area occupata.
    //
    if (! head->allocated)
      {
        printf ("[%s] ERROR: cannot re-allocate %08X that is "
                "not already allocated!", __func__, (uintptr_t) ptr);
      }
    //
    // Arrotonda in eccesso il valore di «size», in modo che sia un
    // multiplo della dimensione di «mm_head_t». Altrimenti, la
    // collocazione dei blocchi successivi può avvenire in modo
    // non allineato.
    //
    size = (size + (sizeof (mm_head_t)) -1);
    size = size / (sizeof (mm_head_t));
    size = size * (sizeof (mm_head_t));
    //
    // Determina la dimensione attuale.
    //
    if ((head->next * (sizeof (mm_head_t))) == start)
      {
        actual_size = os.mem_ph.available_e - ((uintptr_t) ptr);
      }
    else
      {
        actual_size = (head->next * (sizeof (mm_head_t))) - ((uintptr_t) ptr);
      }
    //
    // Se la dimensione richiesta è inferiore, può ridurre
    // l'estensione del blocco.
    //
    if (size == actual_size)
      {
        return ptr;
      }
    else if (size <= (actual_size - (sizeof (mm_head_t)) * 2))
      {
        //
        // Si può ricavare lo spazio libero rimanente.
        //
        head_new = (mm_head_t *) (((char *) ptr) + size);
        //
        head_new->next = head->next;
        head_new->allocated=0;
        //
        head->next = ((uintptr_t) head_new) / (sizeof (mm_head_t));
        //
        return ptr;
      }
    else if (size < actual_size)
      {
        //
        // Anche se è minore, non si può ridurre lo spazio usato
        // effettivamente.
        //
        return ptr;
      }
    else
      {
        //
        // La dimensione richiesta è maggiore.
        //
        ptr_new = malloc (size);
        //
```



```
        if (ptr_new)
          {
            //
            // Ricopia i dati nella nuova collocazione.
            //
            memcpy (ptr_new, ptr, actual_size);
            //
            // Libera la collocazione vecchia.
            //
            free (ptr);
            //
            return ptr_new;
          }
        else
          {
            return NULL;
          }
      }
}
```

Verifica del funzionamento

Per utilizzare le funzioni ***mm_init()*** e ***mm_list()*** occorre aggiornare il file '`kernel_main.c`', aggiungendo in modo particolare delle istruzioni per verificare il funzionamento delle funzioni di allocazione della memoria.

Figura u170.10. Modifiche da apportare al file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
…
        mboot_info (info);
        mboot_show (info);
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        void *p0 = malloc (0x100);
        void *p1 = malloc (0x1000);
        void *p2 = malloc (0x10000);
                  malloc (0x100000);
        p0 = realloc (p0, 0x1000);
        p1 = realloc (p1, 0x100);
        p2 = realloc (p2, 0x100000);
        free (p1);
        mm_list ();
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove si può osservare la mappa della memoria alla fine delle operazioni di allocazione e riallocazione eseguite:

```
 05 20070820133728
 [mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
 [mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
 [kernel_memory_show] kernel 00100000..0010C65C  avail. 0010C65C..01EF0000
 [kernel_memory_show] text   00100000..001039D8  rodata 001039D8..00103CD4
 [kernel_memory_show] data   00103CD4..00103CD4  bss    00103CE0..0010C65C
 [kernel_memory_show] limit  00001EF0
 [gdt_print] base: 0x0010BD28 limit: 0x0017
 [gdt_print] 0 00000000000000000000000000000000 00000000100000000010000000000000
 [gdt_print] 1 00000000000000000000111011110000 00000000110000001001101000000000
 [gdt_print] 2 00000000000000000000111011110000 00000000110000001001001000000000
 [mm_init] available memory: 31340960 byte
 [mm_list] free 0010C660..0010D764 size 00001104
 [mm_list] used 0010D768..0011D768 size 00010000
 [mm_list] used 0011D76C..0021D76C size 00100000
 [mm_list] used 0021D770..0021E770 size 00001000
 [mm_list] used 0021E774..0031E774 size 00100000
 [mm_list] free 0021E778..01EF0000 size 01BD1888
 [kernel_main] system halted
```

Nelle figure successive viene mostrato, schematicamente, ciò che accade. La prima figura mostra lo stato della lista della memoria dopo le prime quattro allocazioni; le figure successive mostrano le riallocazioni che vengono fatte dopo, una a una, per finire con la liberazione della zona associata alla variabile ***p1***. Purtroppo, nelle figure non è stato possibile usare delle proporzioni realistiche.

Figura u170.12. La lista della memoria dopo le prime quattro allocazioni.

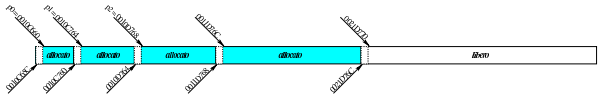


Figura u170.13. La lista della memoria dopo la riallocazione di ***p0***.

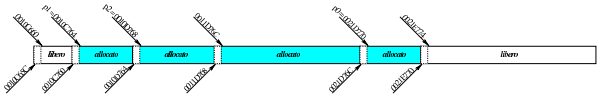


Figura u170.14. La lista della memoria dopo la riallocazione di ***p1***.

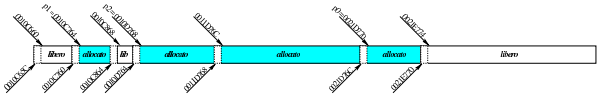


Figura u170.15. La lista della memoria dopo la riallocazione di ***p2***.

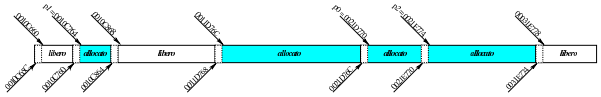


Figura u170.16. La lista della memoria dopo l'eliminazione di ***p1***.

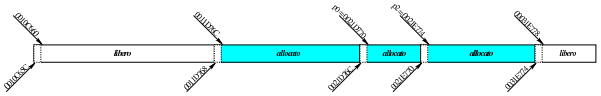


Tabella IDT

File di intestazione «int.h» e file delle routine di interruzione «isr.s» 1955
Funzioni per definire la tabella IDT 1960
Gestione delle interruzioni 1963
Piccole funzioni di contorno 1965
Verifica del funzionamento 1965

`cli.s` 1965 `idt_desc_int.c` 1960 `idt_load.c` 1960 `idt_print.c` 1960 `int.h` 1955 `irq_remap.c` 1960 `isr.s` 1955 `isr_exception_name.c` 1963 `isr_exception_unrecoverable.c` 1963 `isr_irq.c` 1963 `isr_syscall.c` 1963 `sti.s` 1965

In questa fase dello sviluppo del sistema è opportuno predisporre la tabella IDT (*interrupt description table*), con le eccezioni del microprocessore e le interruzioni hardware (IRQ), anche se inizialmente nulla viene gestito effettivamente.

File di intestazione «int.h» e file delle routine di interruzione «isr.s»

Il file di intestazione '`int.h`' contiene la dichiarazione delle funzioni per la gestione delle interruzioni. Viene proposto subito nella sua versione completa, anche se non tutte le funzioni dichiarate vengono presentate immediatamente.

Listato u171.1. '`./05/include/kernel/int.h`'

```
#ifndef _INT_H
#define _INT_H  1

#include <inttypes.h>
#include <stdbool.h>
#include <stdarg.h>
#include <kernel/os.h>

void idt_desc_int    (int      desc,
                      uint32_t offset,
                      uint16_t selector,
                      bool     present,
                      char     type,
                      char     dpl);

void  idt_load       (void *idtr);
void  idt            (void);
void  irq_remap      (unsigned int offset_1, unsigned int offset_2);
char *exception_name (int exception);
void  idt_print      (void *idtr);

void isr_0  (void);
void isr_1  (void);
void isr_2  (void);
void isr_3  (void);
void isr_4  (void);
void isr_5  (void);
void isr_6  (void);
void isr_7  (void);
void isr_8  (void);
void isr_9  (void);
void isr_10 (void);
void isr_11 (void);
void isr_12 (void);
void isr_13 (void);
void isr_14 (void);
void isr_15 (void);
void isr_16 (void);
void isr_17 (void);
void isr_18 (void);
void isr_19 (void);
void isr_20 (void);
void isr_21 (void);
void isr_22 (void);
void isr_23 (void);
void isr_24 (void);
void isr_25 (void);
void isr_26 (void);
void isr_27 (void);
void isr_28 (void);
void isr_29 (void);
void isr_30 (void);
void isr_31 (void);
void isr_32 (void);
void isr_33 (void);
void isr_34 (void);
void isr_35 (void);
void isr_36 (void);
void isr_37 (void);
void isr_38 (void);
```

```
void isr_39 (void);
void isr_40 (void);
void isr_41 (void);
void isr_42 (void);
void isr_43 (void);
void isr_44 (void);
void isr_45 (void);
void isr_46 (void);
void isr_47 (void);
void isr_128 (void);

void sti (void);
void cli (void);

void isr_exception_unrecoverable (uint32_t eax, uint32_t ecx, uint32_t edx,
                                  uint32_t ebx, uint32_t ebp, uint32_t esi,
                                  uint32_t edi, uint32_t ds,  uint32_t es,
                                  uint32_t fs,  uint32_t gs,
                                  uint32_t interrupt, uint32_t error,
                                  uint32_t eip, uint32_t cs, uint32_t eflags);

void isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx, uint32_t ebx,
              uint32_t ebp, uint32_t esi, uint32_t edi, uint32_t ds,
              uint32_t es,  uint32_t fs,  uint32_t gs,  uint32_t interrupt);

uint32_t isr_syscall (uint32_t start, ...);
uint32_t int_128     (void);

#endif
```

Si può osservare l'elenco delle funzioni '**`isr_n()`**', per la gestione delle varie interruzioni catalogate nella tabella IDT. In particolare, l'interruzione 128_{10}, ovvero 80_{16}, viene usata per le chiamate di sistema. Queste funzioni sono dichiarate formalmente nel file '`isr.s`' che viene mostrato integralmente nel listato successivo.

Listato u171.2. '`./05/lib/int/isr.s`'

```
.extern isr_exception_unrecoverable
.extern isr_irq
.extern isr_syscall
#
.globl isr_0
.globl isr_1
.globl isr_2
.globl isr_3
.globl isr_4
.globl isr_5
.globl isr_6
.globl isr_7
.globl isr_8
.globl isr_9
.globl isr_10
.globl isr_11
.globl isr_12
.globl isr_13
.globl isr_14
.globl isr_15
.globl isr_16
.globl isr_17
.globl isr_18
.globl isr_19
.globl isr_20
.globl isr_21
.globl isr_22
.globl isr_23
.globl isr_24
.globl isr_25
.globl isr_26
.globl isr_27
.globl isr_28
.globl isr_29
.globl isr_30
.globl isr_31
.globl isr_32
.globl isr_33
.globl isr_34
.globl isr_35
.globl isr_36
.globl isr_37
.globl isr_38
.globl isr_39
.globl isr_40
.globl isr_41
.globl isr_42
.globl isr_43
.globl isr_44
.globl isr_45
.globl isr_46
.globl isr_47
.globl isr_128

    ###############################################################################
    # Nella pila è già stato inserito dal microprocessore:                        #
    #   [omissis]                                                                 #
    #   push %eflags                                                              #
    #   push %cs                                                                  #
    #   push %eip                                                                 #
    ###############################################################################

isr_0:          # «division by zero exception»
    cli
    push $0     # Codice di errore fittizio.
    push $0     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_1:          # «debug exception»
    cli
    push $0     # Codice di errore fittizio.
    push $1     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_2:          # «non maskable interrupt exception»
    cli
    push $0     # Codice di errore fittizio.
    push $2     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_3:          # «breakpoint exception»
    cli
    push $0     # Codice di errore fittizio.
    push $3     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_4:          # «into detected overflow exception»
    cli
    push $0     # Codice di errore fittizio.
    push $4     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_5:          # «out of bounds exception»
    cli
    push $0     # Codice di errore fittizio.
    push $5     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_6:          # «invalid opcode exception»
    cli
    push $0     # Codice di errore fittizio.
    push $6     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_7:          # «no coprocessor exception»
    cli
    push $0     # Codice di errore fittizio.
    push $7     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_8:          # «double fault exception»
    cli
    #
    push $8     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_9:          # «coprocessor segment overrun exception»
    cli
    push $0     # Codice di errore fittizio.
    push $9     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_10:         # «bad TSS exception»
    cli
    #
    push $10    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_11:         # «segment not present exception»
    cli
    #
    push $11    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_12:         # «stack fault exception»
    cli
    #
    push $12    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_13:         # «general protection fault exception»
    cli
    #
    push $13    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_14:         # «page fault exception»
    cli
    #
    push $14    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_15:         # «unknown interrupt exception»
    cli
    push $0     # Codice di errore fittizio.
    push $15    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_16:         # «coprocessor fault exception»
    cli
    push $0     # Codice di errore fittizio.
    push $16    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_17:         # «alignment check exception»
    cli
```

```
    push $0     # Codice di errore fittizio.
    push $17    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_18:         # «machine check exception»
    cli
    push $0     # Codice di errore fittizio.
    push $18    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_19:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $19    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_20:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $20    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_21:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $21    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_22:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $22    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_23:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $23    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_24:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $24    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_25:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $25    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_26:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $26    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_27:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $27    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_28:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $28    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_29:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $29    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_30:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $30    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_31:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $31    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_32:         # IRQ 0: «timer»
    cli
    push $0     # Codice di errore fittizio.
    push $32    # Numero IRQ + 32.
    jmp irq
#
isr_33:         # IRQ 1: tastiera
    cli
    push $0     # Codice di errore fittizio.
    push $33    # Numero IRQ + 32.
    jmp irq
#
isr_34:         # IRQ 2: viene attivato per gli IRQ da 8 a 15.
    cli
    push $0     # Codice di errore fittizio.
```



```
    push $34    # Numero IRQ + 32.
    jmp irq
#
isr_35:         # IRQ 3
    cli
    push $0     # Codice di errore fittizio.
    push $35    # Numero IRQ + 32.
    jmp irq
#
isr_36:         # IRQ 4
    cli
    push $0     # Codice di errore fittizio.
    push $36    # Numero IRQ + 32.
    jmp irq
#
isr_37:         # IRQ 5
    cli
    push $0     # Codice di errore fittizio.
    push $37    # Numero IRQ + 32.
    jmp irq
#
isr_38:         # IRQ 6: unità a dischetti
    cli
    push $0     # Codice di errore fittizio.
    push $38    # Numero IRQ + 32.
    jmp irq
#
isr_39:         # IRQ 7: LPT 1
    cli
    push $0     # Codice di errore fittizio.
    push $39    # Numero IRQ + 32.
    jmp irq
#
isr_40:         # IRQ 8: «real time clock (RTC)»
    cli
    push $0     # Codice di errore fittizio.
    push $40    # Numero IRQ + 32.
    jmp irq
#
isr_41:         # IRQ 9
    cli
    push $0     # Codice di errore fittizio.
    push $41    # Numero IRQ + 32.
    jmp irq
#
isr_42:         # IRQ 10
    cli
    push $0     # Codice di errore fittizio.
    push $42    # Numero IRQ + 32.
    jmp irq
#
isr_43:         # IRQ 11
    cli
    push $0     # Codice di errore fittizio.
    push $43    # Numero IRQ + 32.
    jmp irq
#
isr_44:         # IRQ 12: mouse PS/2
    cli
    push $0     # Codice di errore fittizio.
    push $44    # Numero IRQ + 32.
    jmp irq
#
isr_45:         # IRQ 13: coprocessore matematico
    cli
    push $0     # Codice di errore fittizio.
    push $45    # Numero IRQ + 32.
    jmp irq
#
isr_46:         # IRQ 14: canale IDE primario
    cli
    push $0     # Codice di errore fittizio.
    push $46    # Numero IRQ + 32.
    jmp irq
#
isr_47:         # IRQ 15: canale IDE secondario
    cli
    push $0     # Codice di errore fittizio.
    push $47    # Numero IRQ + 32.
    jmp irq
#
isr_128:        # Chiamate di sistema.
    cli
    call isr_syscall
    iret
#
# Eccezioni che per il momento non sono gestibili.
#
exception_unrecoverable:

    ###########################################################################
    # A questo punto, nella pila sono stati aggiunti:                         #
    #   push $<n_errore>                                                      #
    #   push $<n_voce_idt>                                                    #
    ###########################################################################

    pushl %gs
    pushl %fs
    pushl %es
    pushl %ds
    pushl %edi
    pushl %esi
    pushl %ebp
```

```
    pushl %ebx
    pushl %edx
    pushl %ecx
    pushl %eax
    #
    call isr_exception_unrecoverable
    #
    popl %eax
    popl %ecx
    popl %edx
    popl %ebx
    popl %ebp
    popl %esi
    popl %edi
    popl %ds
    popl %es
    popl %fs
    popl %gs
    add $4, %esp        # espelle il numero dell'eccezione
    add $4, %esp        # espelle il codice di errore
    #
    iret
#
# IRQ hardware.
#
irq:

    ###############################################################
    # A questo punto, nella pila sono stati aggiunti:             #
    #   push $0                                                   #
    #   push $<n_voce_idt>                                        #
    ###############################################################

    pushl %gs
    pushl %fs
    pushl %es
    pushl %ds
    pushl %edi
    pushl %esi
    pushl %ebp
    pushl %ebx
    pushl %edx
    pushl %ecx
    pushl %eax
    #
    call isr_irq
    #
    popl %eax
    popl %ecx
    popl %edx
    popl %ebx
    popl %ebp
    popl %esi
    popl %edi
    popl %ds
    popl %es
    popl %fs
    popl %gs
    add $4, %esp        # espelle il numero dell'interruzione
    add $4, %esp        # espelle il codice di errore fittizio.
    #
    iret
#
```

Funzioni per definire la tabella IDT

Per facilitare la compilazione della tabella IDT viene usata la funzione ***idt_desc_int()***, con la quale si deve specificare il numero del descrittore della tabella e i dati da inserirvi. La tabella IDT è definita nella variabile strutturata ***os.idt***, dichiarata nel file 'os.h'.

Listato u171.3. './05/lib/int/idt_desc_int.c'

```
#include <kernel/int.h>
void
idt_desc_int (int      desc,
              uint32_t offset,
              uint16_t selector,
              bool     present,
              char     type,
              char     dpl)
{
    //
    // Azzera i bit riservati e quello di sistema.
    //
    os.idt[desc].filler = 0;
    os.idt[desc].system = 0;
    //
    // Indirizzo relativo.
    //
    os.idt[desc].offset_a = (offset & 0x0000FFFF);
    os.idt[desc].offset_b = (offset / 0x10000);
    //
    // Selettore.
    //
    os.idt[desc].selector = selector;
    //
    // Voce valida o meno.
    //
    os.idt[desc].present = present;
    //
    // Tipo (gate type).
    //
    os.idt[desc].type = (type & 0x0F);
    //
    // DPL.
    //
    os.idt[desc].dpl = (dpl & 0x03);
}
```

Per verificare il contenuto della tabella IDT viene predisposta la funzione ***idt_print()*** che richiede come parametro il puntatore all'area di memoria che descrive il registro ***IDTR***. Così come viene proposta, la funzione mostra il contenuto completo della tabella IDT, ma questo supera generalmente le righe visualizzabili sullo schermo; pertanto, in caso di necessità, la funzione va modificata in modo da mostrare solo la porzione di proprio interesse.

Listato u171.4. './05/lib/int/idt_print.c'

```
#include <kernel/int.h>
#include <stdio.h>
//
// Mostra il contenuto di una tabella IDT, a partire dal puntatore al
// registro IDTR in memoria. Pertanto non si avvale, volutamente, della
// struttura già predisposta con il linguaggio C, mentre «local_idtr_t»
// viene creata qui solo provvisoriamente, per uso interno. Ciò serve ad
// assicurare che questa funzione compia il proprio lavoro in modo
// indipendente, garantendo la visualizzazione di dati reali.
//
typedef struct {
    uint16_t  limit;
    uint32_t  base;
} __attribute__ ((packed)) local_idtr_t;
//
void
idt_print (void *idtr)
{
    local_idtr_t *g = idtr;
        uint32_t *p = (uint32_t *) g->base;

    int max = (g->limit + 1) / (sizeof (uint32_t));
    int i;

    for (i = 0; i < max; i+=2)
      {
        printf ("[%s] %02" PRIx32 " %08X %08X\n",
                __func__, i/2, p[i], p[i+1]);
      }
}
```

La funzione ***irq_remap()*** è necessaria per rimappare le interruzioni hardware nella tabella IDT, in modo che non intralcino quelle associate alle eccezioni. La funzione richiede l'indicazione del numero iniziale di interruzione per i due gruppi di IRQ (da IRQ 0 a IRQ 7 e da IRQ 8 a IRQ 15). Successivamente, nella funzione ***idt()***, viene usata ***irq_remap()*** in modo da rimappare le interruzioni hardware a partire da 32, per finire a 47.

Listato u171.5. './05/lib/int/irq_remap.c'

```
#include <kernel/int.h>
#include <stdio.h>
void
irq_remap (unsigned int offset_1, unsigned int offset_2)
{
    //
    // PIC_P è il PIC primario o «master»;
    // PIC_S è il PIC secondario o «slave».
    //
    // Quando si manifesta un IRQ che riguarda il PIC secondario,
    // il PIC primario riceve IRQ 2.
    //
    // ICW = initialization command word.
    // OCW = operation command word.
    //
    printf ("[%s] PIC (programmable interrupt controller) remap: ", __func__);

    outb (0x20, 0x10 + 0x01);   // Inizializzazione: 0x10 significa che
    outb (0xA0, 0x10 + 0x01);   // si tratta di ICW1; 0x01 significa che
    printf ("ICW1");            // si deve arrivare fino a ICW4.

    outb (0x21, offset_1);      // ICW2: PIC_P a partire da «offset_1».
    outb (0xA1, offset_2);      //       PIC_S a partire da «offset_2».
```

```
    printf (", ICW2");
    outb (0x21, 0x04);          // ICW3 PIC_P: IRQ2 pilotato da PIC_S.
    outb (0xA1, 0x02);          // ICW3 PIC_S: pilota IRQ2 di PIC_P.
    printf (", ICW3");
    outb (0x21, 0x01);          // ICW4: si precisa solo la modalità
    outb (0xA1, 0x01);          // del microprocessore; 0x01 = 8086.
    printf (", ICW4");

    outb (0x21, 0x00);          // OCW1: azzera la maschera in modo da
    outb (0xA1, 0x00);          // abilitare tutti i numeri IRQ.
    printf (", OCW1.\n");
}
```

Per caricare la tabella IDT dichiarata in memoria, occorre predisporre la copia del registro ***IDTR*** con i riferimenti necessari a raggiungerla, quindi va usata l'istruzione '**`LIDT`**', con il linguaggio assemblatore. La funzione ***idt_load()*** viene usata per pilotare l'istruzione '**`LIDT`**'.

Listato u171.6. '`./05/lib/int/idt_load.s`'

```
.globl  idt_load
#
idt_load:
    enter $0, $0
    .equ idtr_pointer,  8          # Primo argomento.
    mov  idtr_pointer(%ebp), %eax  # Copia il puntatore
                                   # in EAX.
    leave
    #
    lidt (%eax)   # Utilizza la tabella IDT a cui punta EAX.
    #
    ret
```

La funzione ***idt()*** utilizza le altre descritte in questa sezione, per mettere in funzione la gestione delle interruzioni.

Listato u171.7. '`./05/lib/int/idt.c`'

```
#include <kernel/int.h>
void
idt (void)
{
    //
    // Imposta i dati necessari al registro IDTR.
    //
    os.idtr.limit = (sizeof (os.idt) - 1);
    os.idtr.base  = (uint32_t) &os.idt[0];
    //
    // Azzera le voci previste dell'array «os.idt[]».
    //
    int i;
    for (i = 0; i < ((sizeof (os.idt)) / 8); i++)
      {
        idt_desc_int (i, 0, 0, 0, 0, 0);
      }
    //
    // Associa le interruzioni hardware da IRQ 0 a IRQ 7
    // a partire dal descrittore 32 e quelle da IRQ 8 a
    // IRQ 15, a partire dal descrittore 40.
    //
    irq_remap (32, 40);
    //
    // Associa le routine ISR ai descrittori della tabella
    // IDT.
    //
    idt_desc_int ( 0, (uint32_t) isr_0,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 1, (uint32_t) isr_1,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 2, (uint32_t) isr_2,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 3, (uint32_t) isr_3,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 4, (uint32_t) isr_4,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 5, (uint32_t) isr_5,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 6, (uint32_t) isr_6,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 7, (uint32_t) isr_7,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 8, (uint32_t) isr_8,  0x0008, 1, 0xE, 0);
    idt_desc_int ( 9, (uint32_t) isr_9,  0x0008, 1, 0xE, 0);
    idt_desc_int (10, (uint32_t) isr_10, 0x0008, 1, 0xE, 0);
    idt_desc_int (11, (uint32_t) isr_11, 0x0008, 1, 0xE, 0);
    idt_desc_int (12, (uint32_t) isr_12, 0x0008, 1, 0xE, 0);
    idt_desc_int (13, (uint32_t) isr_13, 0x0008, 1, 0xE, 0);
    idt_desc_int (14, (uint32_t) isr_14, 0x0008, 1, 0xE, 0);
    idt_desc_int (15, (uint32_t) isr_15, 0x0008, 1, 0xE, 0);
    idt_desc_int (16, (uint32_t) isr_16, 0x0008, 1, 0xE, 0);
    idt_desc_int (17, (uint32_t) isr_17, 0x0008, 1, 0xE, 0);
```



```
    idt_desc_int (18, (uint32_t) isr_18, 0x0008, 1, 0xE, 0);
    idt_desc_int (19, (uint32_t) isr_19, 0x0008, 1, 0xE, 0);
    idt_desc_int (20, (uint32_t) isr_20, 0x0008, 1, 0xE, 0);
    idt_desc_int (21, (uint32_t) isr_21, 0x0008, 1, 0xE, 0);
    idt_desc_int (22, (uint32_t) isr_22, 0x0008, 1, 0xE, 0);
    idt_desc_int (23, (uint32_t) isr_23, 0x0008, 1, 0xE, 0);
    idt_desc_int (24, (uint32_t) isr_24, 0x0008, 1, 0xE, 0);
    idt_desc_int (25, (uint32_t) isr_25, 0x0008, 1, 0xE, 0);
    idt_desc_int (26, (uint32_t) isr_26, 0x0008, 1, 0xE, 0);
    idt_desc_int (27, (uint32_t) isr_27, 0x0008, 1, 0xE, 0);
    idt_desc_int (28, (uint32_t) isr_28, 0x0008, 1, 0xE, 0);
    idt_desc_int (29, (uint32_t) isr_29, 0x0008, 1, 0xE, 0);
    idt_desc_int (30, (uint32_t) isr_10, 0x0008, 1, 0xE, 0);
    idt_desc_int (31, (uint32_t) isr_31, 0x0008, 1, 0xE, 0);
    idt_desc_int (32, (uint32_t) isr_32, 0x0008, 1, 0xE, 0);
    idt_desc_int (33, (uint32_t) isr_33, 0x0008, 1, 0xE, 0);
    idt_desc_int (34, (uint32_t) isr_34, 0x0008, 1, 0xE, 0);
    idt_desc_int (35, (uint32_t) isr_35, 0x0008, 1, 0xE, 0);
    idt_desc_int (36, (uint32_t) isr_36, 0x0008, 1, 0xE, 0);
    idt_desc_int (37, (uint32_t) isr_37, 0x0008, 1, 0xE, 0);
    idt_desc_int (38, (uint32_t) isr_38, 0x0008, 1, 0xE, 0);
    idt_desc_int (39, (uint32_t) isr_39, 0x0008, 1, 0xE, 0);
    idt_desc_int (40, (uint32_t) isr_40, 0x0008, 1, 0xE, 0);
    idt_desc_int (41, (uint32_t) isr_41, 0x0008, 1, 0xE, 0);
    idt_desc_int (42, (uint32_t) isr_42, 0x0008, 1, 0xE, 0);
    idt_desc_int (43, (uint32_t) isr_43, 0x0008, 1, 0xE, 0);
    idt_desc_int (44, (uint32_t) isr_44, 0x0008, 1, 0xE, 0);
    idt_desc_int (45, (uint32_t) isr_45, 0x0008, 1, 0xE, 0);
    idt_desc_int (46, (uint32_t) isr_46, 0x0008, 1, 0xE, 0);
    idt_desc_int (47, (uint32_t) isr_47, 0x0008, 1, 0xE, 0);
    //
    // Questo è per le chiamate di sistema.
    //
    idt_desc_int (128, (uint32_t) isr_128, 0x0008, 1, 0xE,
                  0);
    //
    // Rende operativa la tabella con le eccezioni e gli
    // IRQ.
    //
    idt_load (&os.idtr);
    //
    // Abilita le interruzioni hardware (IRQ).
    //
    sti ();
}
```

Gestione delle interruzioni

Le funzioni '**`isr_n()`**' si limitano a chiamare altre funzioni scritte in linguaggio C, per la gestione delle eccezioni, delle interruzioni hardware e per le chiamate di sistema. In questa fase vengono mostrate le funzioni per la gestione delle eccezioni, anche se in forma estremamente limitata, e si propongono temporaneamente delle funzioni fittizie per la gestione degli altri casi.

Listato u171.8. '`./05/lib/int/isr_exception_unrecoverable.c`'

```
#include <kernel/int.h>
#include <stdio.h>
void
isr_exception_unrecoverable (uint32_t eax, uint32_t ecx,
                             uint32_t edx, uint32_t ebx,
                             uint32_t ebp, uint32_t esi,
                             uint32_t edi, uint32_t ds,
                             uint32_t es, uint32_t fs,
                             uint32_t gs,
                             uint32_t interrupt,
                             uint32_t error, uint32_t eip,
                             uint32_t cs,
                             uint32_t eflags)
{
    printf ("[%s] ERROR: exception %i: \"%s\"\n",
            __func__, interrupt,
            exception_name (interrupt));
    //
    _Exit (0);
}
```

La funzione ***isr_exception_unrecoverable()***, appena mostrata, vie-

ne chiamata dal file ‘isr.s’, per le interruzioni che riguardano le eccezioni. La funzione si limita a visualizzare un messaggio di errore e a fermare il sistema. Per visualizzare il tipo di eccezione che si è verificato si avvale della funzione ***exception_name()*** che appare nel listato successivo.

Listato u171.9. ‘./05/lib/int/exception_name.c’

```
#include <kernel/int.h>
char
*exception_name (int exception)
{
    char *description[19] = {"division by zero",
                             "debug",
                             "non maskable interrupt",
                             "breakpoint",
                             "into detected overflow",
                             "out of bounds",
                             "invalid opcode",
                             "no coprocessor",
                             "double fault",
                             "coprocessor segmento overrun",
                             "bad TSS",
                             "segment not present",
                             "stack fault",
                             "general protection fault",
                             "page fault",
                             "unknown interrupt",
                             "coprocessor fault",
                             "alignment check",
                             "machine check"};
    //
    if (exception >= 0 && exception <= 18)
      {
        return description[exception];
      }
    else
      {
        return "unknown";
      }
}
```

A proposito della funzione ***exception_name()*** va osservata la particolarità del comportamento del compilatore GNU C, il quale utilizza, senza che ciò sia stato richiesto espressamente, la funzione standard ***memcpy()***. Pertanto, tale funzione deve essere disponibile, altrimenti, in fase di collegamento (*link*) la compilazione fallisce.

Per la gestione delle interruzioni hardware è competente la funzione ***isr_irq()***, ma per il momento viene proposta una versione provvisoria, priva di alcuna gestione, dove ci si limita a inviare il messaggio «EOI» ai PIC (*programmable interrupt controller*) coinvolti.

Listato u171.10. Una prima versione del file ‘./05/lib/int/isr_irq.c’

```
#include <kernel/int.h>
#include <kernel/io.h>
void
isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx,
         uint32_t ebx, uint32_t ebp, uint32_t esi,
         uint32_t edi, uint32_t ds, uint32_t es,
         uint32_t fs,  uint32_t gs,  uint32_t interrupt)
{
    int irq = interrupt - 32;
    //
    // Finito il compito della funzione che deve reagire
    // all'interruzione IRQ, occorre informare i PIC
    // (programmable interrupt controller).
    //
    // Se il numero IRQ è tra 8 e 15, manda un messaggio
    // «EOI»
    // (End of IRQ) al PIC 2.
    //
    if (irq >= 8)
      {
        outb (0xA0, 0x20);
      }
    //
    // Poi manda un messaggio «EOI» al PIC 1.
    //
    outb (0x20, 0x20);
}
```


Anche la funzione ***isr_syscall()*** che dovrebbe prendersi cura delle chiamate di sistema, viene proposta inizialmente priva di alcun effetto.

Listato u171.11. Una prima versione del file ‘./05/lib/int/isr_syscall.c’

```
#include <kernel/int.h>
uint32_t
isr_syscall (uint32_t start, ...)
{
    return 0;
}
```

Piccole funzioni di contorno

Per facilitare l'accesso alle istruzioni ‘**STI**’ e ‘**CLI**’ del linguaggio assemblatore, vengono predisposte due funzioni con lo stesso nome.

Listato u171.12. ‘./05/lib/int/cli.s’

```
.globl cli
#
cli:
    cli
    ret
```

Listato u171.13. ‘./05/lib/int/sti.s’

```
.globl sti
#
sti:
    sti
    ret
```

Verifica del funzionamento

Per verificare il lavoro svolto fino a questo punto, è necessario sviluppare ulteriormente i file ‘kernel_main.c’, dove in particolare si va a produrre un errore che causa un eccezione dovuta a una divisione per zero.

Figura u171.14. Modifiche da apportare al file ‘./05/kernel/kernel_main.c’

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
…
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Omissis.
        //
        //
        // Predispone la tabella IDT.
        //
        idt();
        //
        // Crea un errore volontario.
        //
        int x = 3;
        x = 7 / (x - 3);        // x = 7 / 0
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove alla fine si vede la segnalazione di errore dovuta alla divisione per zero:

```
05 20070821144531
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010D8BC  avail. 0010D8BC..01EF0000
[kernel_memory_show] text   00100000..001049DC  rodata 001049E0..00104F34
[kernel_memory_show] data   00104F34..00104F34  bss    00104F40..0010D8BC
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010CF88 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000100000000010000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[mm_init] available memory: 31336256 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
[isr_exception_unrecoverable] ERROR: exception 0: "division by zero"
```


Chiamate di sistema

File di intestazione «syscall.h» 1967
Fasi successive all'interruzione 1967
Verifica del funzionamento 1968

`int_128.s` 1967 `isr_syscall.c` 1967 `syscall.c` 1967 `syscall.h` 1967 `vsyscall.c` 1967

Nel sistema in corso di realizzazione sono previste le chiamate di sistema, anche se in pratica sono inutili, dal momento che non è possibile gestire processi elaborativi indipendenti. Queste chiamate si ottengono mettendo gli argomenti nella pila e utilizzando l'interruzione 128 (ovvero 80_{16}). Si osservi che questo meccanismo è diverso da quello usato dal kernel Linux, dove gli argomenti sono passati normalmente attraverso i registri del microprocessore.

Il punto di inizio per una chiamata di sistema è la funzione ***syscall()***, con la quale va indicato il numero della chiamata, seguito dagli argomenti necessari, in base al contesto.

Listato u172.1. '`./05/lib/sys/syscall.c`'

```
#include <sys/syscall.h>
#include <kernel/int.h>
uint32_t
syscall (int n, ...)
{
    return int_128 ();
}
```

Come si vede, ci si limita a utilizzare la funzione ***int_128()***, scritta però in linguaggio assemblatore, come si vede nel listato successivo.

Listato u172.2. '`./05/lib/int/int_128.s`'

```
.globl int_128
#
int_128:
    int $128
    ret
```

Questa doppia mediazione ha delle conseguenze nella composizione della pila dei dati, al momento dell'avvio della funzione che deve trattare l'interruzione.

File di intestazione «syscall.h»

Il file di intestazione '`syscall.h`' dichiara le funzioni usate per generare una chiamata di sistema e poi per eseguirla; inoltre, si definiscono delle macro-variabili per dare un nome alle chiamate che in realtà sono indicate solo per numero.

Listato u172.3. '`./05/include/sys/syscall.h`'

```
#ifndef _SYSCALL_H
#define _SYSCALL_H      1

#include <inttypes.h>
#include <stdarg.h>

#define SYSCALL_malloc       1
#define SYSCALL_realloc      2
#define SYSCALL_free         3
#define SYSCALL_console_putc 4

uint32_t syscall  (int n, ...);
uint32_t vsyscall (int n, va_list ap);

#endif
```

Fasi successive all'interruzione

Una volta provocata l'interruzione 128, si ottiene l'attivazione della funzione ***isr_128()***, la quale avvia a sua volta la funzione ***isr_syscall()*** che deve provvedere a ripescare gli argomenti della chiamata originale, quindi avvia la funzione che può elaborarli: ***vsyscall()***.

Listato u172.4. '`./05/lib/int/isr_syscall.c`'

```
#include <kernel/int.h>
#include <sys/syscall.h>
uint32_t
isr_syscall (uint32_t start, ...)
{
```

```
    va_list ap;
    uint32_t value;
    //
    // Colloca il puntatore all'inizio.
    //
    va_start (ap, start);
    //
    // Salta i dati che non servono.
    //
    value = va_arg (ap, uint32_t); // CS
    value = va_arg (ap, uint32_t); // EFLAGS
    value = va_arg (ap, uint32_t); // ???
    value = va_arg (ap, uint32_t); // ESP
    value = va_arg (ap, uint32_t); // SS
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // n. chiamata
    //
    // Attualmente «ap» punta all'argomento successivo
    // al numero di chiamata.
    //
    return vsyscall (value, ap);
}
```

Listato u172.5. '`./05/lib/sys/vsyscall.c`'

```
#include <sys/syscall.h>
#include <stdint.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdarg.h>
#include <kernel/vga.h>

uint32_t
vsyscall (int n, va_list ap)
{
    if (n == SYSCALL_malloc)
      {
        size_t  size = va_arg (ap, size_t);
        return (uint32_t) malloc (size);
      }
    else if (n == SYSCALL_realloc)
      {
        void   *ptr  = va_arg (ap, void*);   // Qui, «void*» va scritto
        size_t  size = va_arg (ap, size_t);  // attaccato e senza parentesi.
        return (uint32_t) realloc (ptr, size);
      }
    else if (n == SYSCALL_free)
      {
        void   *ptr  = va_arg (ap, void*);
        free (ptr);
        return 0;
      }
    else if (n == SYSCALL_console_putc)
      {
        int     c   = va_arg (ap, int);
        vga_putc (c);
        return (uint32_t) c;
      }
    else
      {
        printf ("[%s] ERROR: unknown syscall: %i!\n", __func__, n);
        return -1;
      }
}
```

Verifica del funzionamento

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file '`kernel_main.c`' nel modo seguente, allo scopo di visualizzare sullo schermo la parola «Ciao».

Figura u172.6. Modifiche da apportare al file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
…
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Omissis.
        //
        //
        // Predispone la tabella IDT.
        //
        idt();
        //
        // Prova le chiamate di sistema.
        //
        syscall (SYSCALL_console_putc, 'C');
        syscall (SYSCALL_console_putc, 'i');
        syscall (SYSCALL_console_putc, 'a');
        syscall (SYSCALL_console_putc, 'o');
        syscall (SYSCALL_console_putc, '\n');
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove prima della conclusione si vede l'emissione della parola «Ciao»:

```
05 20070821155848
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010DADC  avail. 0010DADC..01EF0000
[kernel_memory_show] text   00100000..00104BA8  rodata 00104BC0..00105144
[kernel_memory_show] data   00105144..00105144  bss    00105160..0010DADC
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010D1A8 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[mm_init] available memory: 31335712 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
Ciao
[kernel_main] system halted
```

Interruzioni hardware

Gestione del temporizzatore 1971
Gestione della tastiera 1972
Verifica del funzionamento 1975

`isr_irq.c` 1971 `keyboard.c` 1972 `keyboard.h` 1972 `keyboard_load.c` 1972 `timer.c` 1971 `timer.h` 1971 `timer_freq.c` 1971

Le interruzioni hardware che vengono gestite in questo sistema sono solo IRQ 0 (temporizzatore o *timer*) e IRQ 1 (tastiera). Il file '`isr_irq.c`' che in precedenza è stato ridotto per sospendere il problema delle interruzioni hardware ha la forma finale del listato successivo.

Listato u173.1. '`./05/lib/int/isr_irq.c`'

```
#include <kernel/int.h>
#include <kernel/io.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
void
isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx, uint32_t ebx,
         uint32_t ebp, uint32_t esi, uint32_t edi, uint32_t ds,
         uint32_t es,  uint32_t fs,  uint32_t gs,  uint32_t interrupt)
{
    int irq = interrupt - 32;
    //
    //
    //
    switch (irq)
      {
        case  0: timer (); break;
        case  1: keyboard (); break;
      }
    //
    // Finito il compito della funzione che deve reagire all'interruzione
    // IRQ, occorre informare i PIC (programmable interrupt controller).
    //
    // Se il numero IRQ è tra 8 e 15, manda un messaggio «EOI»
    // (End of IRQ) al PIC 2.
    //
    if (irq >= 8)
      {
        outb (0xA0, 0x20);
      }
    //
    // Poi manda un messaggio «EOI» al PIC 1.
    //
    outb (0x20, 0x20);
}
```

Gestione del temporizzatore

La gestione del temporizzatore è raccolta dalla libreria che fa capo al file di intestazione '`timer.h`' come appare nel listato successivo.

Listato u173.2. '`./05/include/kernel/timer.h`'

```
#ifndef _TIMER_H
#define _TIMER_H        1

#include <time.h>
#include <kernel/os.h>

void timer      (void);
void timer_freq (clock_t freq);

#endif
```

Il temporizzatore genera impulsi a una frequenza data e a ogni impulso produce un'interruzione. Per regolare tale frequenza occorre comunicare con le porte 43_{16} e 40_{16}, inviando il divisore da applicare alla frequenza di riferimento che è 1,193 181 MHz. La funzione ***timer_freq()*** stabilisce la frequenza da generare, calcolando il divisore da applicare.[1]

Listato u173.3. '`./05/lib/timer/timer_freq.c`'

```
#include <kernel/timer.h>
#include <stdint.h>
#include <stdio.h>
void
timer_freq (clock_t freq)
{
    int input_freq = 1193180;
    //
```

```
    // La frequenza di riferimento è 1,19318 MHz, la quale va
    // divisa per la frequenza che si intende avere effettivamente.
    //
    int divisor = input_freq / freq;
    //
    // Il risultato deve essere un valore intero maggiore di zero
    // e inferiore di UINT16_MAX, altrimenti è stata chiesta una
    // frequenza troppo elevata o troppo bassa.
    //
    if (divisor == 0 || divisor > UINT16_MAX)
      {
        printf ("[%s] ERROR: IRQ 0 frequency wrong: %i Hz!\n"
                "[%s]        The min allowed frequency is 18.22 Hz.\n",
                "[%s]        The max allowed frequency is 1.19 MHz.\n",
                __func__, freq, __func__, __func__);
        return;
      }
    //
    // Il valore che si ottiene, ovvero il «divisore», va
    // comunicato al PIT (programmable interval timer),
    // spezzandolo in due parti.
    //
    outb (0x43, 0x36);
    outb (0x40, divisor & 0x0F);        // Byte inferiore del numero.
    outb (0x40, divisor / 0x10);        // Byte superiore del numero.
    //
    // Annota la frequenza attuale degli impulsi provenienti dal
    // PIT (programmable interval timer).
    //
    os.timer.freq = freq;
}
```

La funzione ***timer()*** è quella che viene eseguita automaticamente, ogni volta che si presenta un'interruzione che deriva da un IRQ 0. Di norma lo scopo di una funzione di questo tipo è controllare la gestione corretta dei processi, ma in mancanza di questi, si potrebbero avviare delle funzioni che assicurano un'esecuzione brevissima, salvo il verificarsi di eventi specifici. Nel listato successivo si presenta una funzione ***timer()*** praticamente vuota e i file di intestazione incorporati sono ipotetici.

Listato u173.4. '`./05/lib/timer/timer.c`'

```
#include <kernel/timer.h>
#include <kernel/int.h>
#include <time.h>
void
timer (void)
{
    //
    // Conta le interruzioni.
    //
    os.timer.clocks++;
    //
    // Dovrebbe lanciare lo «schedulatore», ma qui non c'è;
    // pertanto, lancia direttamente delle applicazioni molto
    // brevi (devono garantire di terminare rapidamente).
    //
    ;
    ;
    ;
}
```

L'incremento della variabile ***os.timer.clocks*** consentirebbe di compiere delle azioni quando risulta trascorso un certo intervallo di tempo. Un'ipotesi di utilizzo potrebbe essere quella seguente, dove, ammesso che la frequenza del temporizzatore sia pari a '**`CLOCKS_PER_SEC`**', al trascorrere di ogni secondo fa qualcosa:

```
void
timer (void)
{
    os.timer.clocks++;
    if ((os.timer.clocks % CLOCKS_PER_SEC) == 0)
      {
        fa_qualcosa
      }
}
```

Gestione della tastiera

La gestione della tastiera è raccolta dalla libreria che fa capo al file di intestazione '`keyboard.h`' come appare nel listato successivo.

Listato u173.6. '`./05/include/kernel/keyboard.h`'

```
#ifndef _KEYBOARD_H
#define _KEYBOARD_H     1

#include <kernel/os.h>

void keyboard (void);
void keyboard_load (void);

#endif
```


La funzione ***keyboard_load()*** definisce una mappa della tastiera, memorizzata negli array ***os.kbd.map1[]*** e ***os.kbd.map2[]***. Le due mappe riguardano i due livelli di scrittura: quello normale e quello che solitamente produce principalmente le maiuscole. L'indice degli array corrisponde al codice grezzo generato dalla tastiera (*scancode*). Il listato successivo riguarda una funzione ***keyboard_load()*** adatta alla disposizione italiana dei simboli, tenendo conto però che non si possono generare lettere accentate.

Listato u173.7. '`./05/lib/keyboard/keyboard_load.c`'

```
#include <kernel/keyboard.h>
void
keyboard_load (void)
{
    int i;
    for (i = 0; i <= 127; i++)
      {
        os.kbd.map1[i] = 0;
        os.kbd.map2[i] = 0;
      }
    //
    //
    //
    os.kbd.map1[1]   = 27;      os.kbd.map2[1]   = 27;
    os.kbd.map1[2]   = '1';     os.kbd.map2[2]   = '!';
    os.kbd.map1[3]   = '2';     os.kbd.map2[3]   = '"';
    os.kbd.map1[4]   = '3';     os.kbd.map2[4]   = 'L';         // 3, £
    os.kbd.map1[5]   = '4';     os.kbd.map2[5]   = '$';
    os.kbd.map1[6]   = '5';     os.kbd.map2[6]   = '%';
    os.kbd.map1[7]   = '6';     os.kbd.map2[7]   = '&';
    os.kbd.map1[8]   = '7';     os.kbd.map2[8]   = '/';
    os.kbd.map1[9]   = '8';     os.kbd.map2[9]   = '(';
    os.kbd.map1[10]  = '9';     os.kbd.map2[10]  = ')';
    os.kbd.map1[11]  = '0';     os.kbd.map2[11]  = '=';
    os.kbd.map1[12]  = '\'';    os.kbd.map2[12]  = '?';
    os.kbd.map1[13]  = 'i';     os.kbd.map2[13]  = '^';         // ì, ^
    os.kbd.map1[14]  = '\b';    os.kbd.map2[14]  = '\b';        // Backspace
    os.kbd.map1[15]  = '\t';    os.kbd.map2[15]  = '\t';
    os.kbd.map1[16]  = 'q';     os.kbd.map2[16]  = 'Q';
    os.kbd.map1[17]  = 'w';     os.kbd.map2[17]  = 'W';
    os.kbd.map1[18]  = 'e';     os.kbd.map2[18]  = 'E';
    os.kbd.map1[19]  = 'r';     os.kbd.map2[19]  = 'R';
    os.kbd.map1[20]  = 't';     os.kbd.map2[20]  = 'T';
    os.kbd.map1[21]  = 'y';     os.kbd.map2[21]  = 'Y';
    os.kbd.map1[22]  = 'u';     os.kbd.map2[22]  = 'U';
    os.kbd.map1[23]  = 'i';     os.kbd.map2[23]  = 'I';
    os.kbd.map1[24]  = 'o';     os.kbd.map2[24]  = 'O';
    os.kbd.map1[25]  = 'p';     os.kbd.map2[25]  = 'P';
    os.kbd.map1[26]  = '[';     os.kbd.map2[26]  = '{';         // è, é
    os.kbd.map1[27]  = ']';     os.kbd.map2[27]  = '}';         // +, *
    os.kbd.map1[28]  = '\n';    os.kbd.map2[28]  = '\n';        // Invio
    os.kbd.map1[30]  = 'a';     os.kbd.map2[30]  = 'A';
    os.kbd.map1[31]  = 's';     os.kbd.map2[31]  = 'S';
    os.kbd.map1[32]  = 'd';     os.kbd.map2[32]  = 'D';
    os.kbd.map1[33]  = 'f';     os.kbd.map2[33]  = 'F';
    os.kbd.map1[34]  = 'g';     os.kbd.map2[34]  = 'G';
    os.kbd.map1[35]  = 'h';     os.kbd.map2[35]  = 'H';
    os.kbd.map1[36]  = 'j';     os.kbd.map2[36]  = 'J';
    os.kbd.map1[37]  = 'k';     os.kbd.map2[37]  = 'K';
    os.kbd.map1[38]  = 'l';     os.kbd.map2[38]  = 'L';
    os.kbd.map1[39]  = '@';     os.kbd.map2[39]  = '@';         // ò, ç
    os.kbd.map1[40]  = '#';     os.kbd.map2[40]  = '#';         // à, °
    os.kbd.map1[41]  = '\\';    os.kbd.map2[41]  = '|';
    os.kbd.map1[43]  = 'u';     os.kbd.map2[43]  = 'U';         // ù, §
    os.kbd.map1[44]  = 'z';     os.kbd.map2[44]  = 'Z';
    os.kbd.map1[45]  = 'x';     os.kbd.map2[45]  = 'X';
    os.kbd.map1[46]  = 'c';     os.kbd.map2[46]  = 'C';
    os.kbd.map1[47]  = 'v';     os.kbd.map2[47]  = 'V';
    os.kbd.map1[48]  = 'b';     os.kbd.map2[48]  = 'B';
    os.kbd.map1[49]  = 'n';     os.kbd.map2[49]  = 'N';
    os.kbd.map1[50]  = 'm';     os.kbd.map2[50]  = 'M';
    os.kbd.map1[51]  = ',';     os.kbd.map2[51]  = ';';
    os.kbd.map1[52]  = '.';     os.kbd.map2[52]  = ':';
    os.kbd.map1[53]  = '-';     os.kbd.map2[53]  = '_';
    os.kbd.map1[56]  = '<';     os.kbd.map2[56]  = '>';
    os.kbd.map1[57]  = ' ';     os.kbd.map2[57]  = ' ';
    //
    os.kbd.map1[55]  = '*';     os.kbd.map2[55]  = '*';
    os.kbd.map1[71]  = '7';     os.kbd.map2[71]  = '7';
    os.kbd.map1[72]  = '8';     os.kbd.map2[72]  = '8';
    os.kbd.map1[73]  = '9';     os.kbd.map2[73]  = '9';
    os.kbd.map1[74]  = '-';     os.kbd.map2[74]  = '-';
    os.kbd.map1[75]  = '4';     os.kbd.map2[75]  = '4';
    os.kbd.map1[76]  = '5';     os.kbd.map2[76]  = '5';
    os.kbd.map1[77]  = '6';     os.kbd.map2[77]  = '6';
    os.kbd.map1[78]  = '+';     os.kbd.map2[78]  = '+';
    os.kbd.map1[79]  = '1';     os.kbd.map2[79]  = '1';
    os.kbd.map1[80]  = '2';     os.kbd.map2[80]  = '2';
    os.kbd.map1[81]  = '3';     os.kbd.map2[81]  = '3';
    os.kbd.map1[82]  = '0';     os.kbd.map2[82]  = '0';
    os.kbd.map1[83]  = '.';     os.kbd.map2[83]  = '.';
    os.kbd.map1[92]  = '/';     os.kbd.map2[92]  = '/';
    os.kbd.map1[96]  = '\n';    os.kbd.map2[96]  = '\n';        // Invio
}
```

La funzione ***keyboard()***, avviata ogni volta che si preme un tasto o lo si rilascia (attraverso l'impulso dato da IRQ 2), interpreta il codice

grezzo proveniente dalla tastiera e aggiorna la variabile strutturata ***os.kbd***. Per esempio tiene traccia della pressione dei tasti [*Ctrl*], [*Alt*] e della selezione delle maiuscole. Quando si tratta di un tasto che deve produrre un carattere, questo viene annotato nella variabile ***os.kbd.key***, ma solo se questa è vuota. In pratica ci deve essere un programma che «consuma» questa informazione, azzerando di conseguenza la variabile. Si osservi che la variabile ***os.kbd.echo***, se contiene un valore diverso da zero, indica la richiesta di visualizzare sullo schermo ciò che si preme ed è controllata dalla macroistruzione ***echo()*** (da ‘`vga.h`’).

Listato u173.8. ‘`./05/lib/keyboard/keyboard.c`’

```
#include <kernel/keyboard.h>
#include <kernel/io.h>
#include <stdio.h>
void
keyboard (void)
{
    unsigned char scancode = inb (0x60);

    //
    // Shift, Shift-Lock, Ctrl, Alt
    //
    switch (scancode)
      {
        case 0x2A: os.kbd.shift = 1; break;
        case 0x36: os.kbd.shift = 1; break;
        case 0xAA: os.kbd.shift = 0; break;
        case 0xB6: os.kbd.shift = 0; break;
        case 0x1D: os.kbd.ctrl  = 1; break;
        case 0x9D: os.kbd.ctrl  = 0; break;
        case 0x38: os.kbd.alt   = 1; break;
        case 0xB8: os.kbd.alt   = 0; break;
        case 0x3A: os.kbd.shift_lock = ! os.kbd.shift_lock; break;
      }
    //
    // Ctrl+
    //
    if (scancode <= 127 && os.kbd.ctrl && os.kbd.key == 0)
      {
        switch (os.kbd.map1[scancode])
          {
            case 'a':  os.kbd.key = 0x01; break;  // SOH
            case 'b':  os.kbd.key = 0x02; break;  // STX
            case 'c':  os.kbd.key = 0x03; break;  // ETX
            case 'd':  os.kbd.key = 0x04; break;  // EOT
            case 'e':  os.kbd.key = 0x05; break;  // ENQ
            case 'f':  os.kbd.key = 0x06; break;  // ACK
            case 'g':  os.kbd.key = 0x07; break;  // BEL
            case 'h':  os.kbd.key = 0x08; break;  // BS
            case 'i':  os.kbd.key = 0x09; break;  // HT
            case 'j':  os.kbd.key = 0x0A; break;  // LF
            case 'k':  os.kbd.key = 0x0B; break;  // VT
            case 'l':  os.kbd.key = 0x0C; break;  // FF
            case 'm':  os.kbd.key = 0x0D; break;  // CR
            case 'n':  os.kbd.key = 0x0E; break;  // SO
            case 'o':  os.kbd.key = 0x0F; break;  // SI
            case 'p':  os.kbd.key = 0x10; break;  // DLE
            case 'q':  os.kbd.key = 0x11; break;  // DC1
            case 'r':  os.kbd.key = 0x12; break;  // DC2
            case 's':  os.kbd.key = 0x13; break;  // DC3
            case 't':  os.kbd.key = 0x14; break;  // DC4
            case 'u':  os.kbd.key = 0x15; break;  // NAK
            case 'v':  os.kbd.key = 0x16; break;  // SYN
            case 'w':  os.kbd.key = 0x17; break;  // ETB
            case 'x':  os.kbd.key = 0x18; break;  // CAN
            case 'y':  os.kbd.key = 0x19; break;  // EM
            case 'z':  os.kbd.key = 0x1A; break;  // SUB
            case '[':  os.kbd.key = 0x1B; break;  // ESC
            case '\\': os.kbd.key = 0x1C; break;  // FS
            case ']':  os.kbd.key = 0x1D; break;  // GS
            case '^':  os.kbd.key = 0x1E; break;  // RS
            case '_':  os.kbd.key = 0x1F; break;  // US
          }
        //
        if (os.kbd.echo && os.kbd.key)
          {
            (void) putchar (os.kbd.key);
          }
      }
    else if (scancode <= 127 && os.kbd.key == 0 && os.kbd.map1[scancode] != 0)
      {
        if (os.kbd.shift || os.kbd.shift_lock)
          {
            os.kbd.key = os.kbd.map2[scancode];
          }
        else
          {
            os.kbd.key = os.kbd.map1[scancode];
          }
        //
        if (os.kbd.echo && os.kbd.key)
          {
            (void) putchar (os.kbd.key);
          }
      }
```



```
}
```

Verifica del funzionamento

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file ‘`kernel_main.c`’ nel modo seguente. Ciò che si ottiene è di poter visualizzare sullo schermo il primo tasto che si preme (ciò avviene subito dopo la dichiarazione che il sistema è arrestato), in quanto non si possono inserire altri caratteri fino a quando «qualcuno» non svuota ***os.kbd.key***.

Figura u173.9. Modifiche da apportare al file ‘`./05/kernel/kernel_main.c`’

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
…
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Predispone il timer.
        //
        timer_freq (CLOCKS_PER_SEC);
        //
        // Predispone la tastiera.
        //
        keyboard_load ();
        echo ();
        //
        // Predispone la tabella IDT.
        //
        idt();
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove dopo l'arresto dichiarato del sistema si può premere un tasto che viene visualizzato:

```
05 20070821183438
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010E45C  avail. 0010E45C..01EF0000
[kernel_memory_show] text   00100000..001054D8  rodata 001054E0..00105AC8
[kernel_memory_show] data   00105AC8..00105AC8  bss    00105AE0..0010E45C
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010DB28 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000100000000100000000000000
[gdt_print] 1 00000000000000000011111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000011111011110000 00000000110000001001001000000000
[mm_init] available memory: 31333280 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
[kernel_main] system halted
h
```

[1] Evidentemente, il limite massimo teorico della frequenza che può essere generata è proprio 1,193181 MHz che si ottiene dividendo semplicemente per uno; inoltre, sapendo che il divisore può avere al massimo il valore 65535, la frequenza minima è di 18,22 Hz, corrispondente al valore predefinito iniziale.

Una specie di «shell»

Realizzazione della shell 1977
Conclusione 1978

`app.h` 1977 `gets.c` 1977 `kernel_main.c` 1978 `shell.c` 1977

Si conclude il lavoro del sistema giocattolo con una shell elementare, la quale deve acquisire i caratteri prodotti dalla tastiera e svolgere un compito in base al comando impartito. Ma prima di vedere il codice della funzione che svolge questo compito è necessario introdurre un'altra funzione, prevista dallo standard, che in precedenza è stata saltata: ***gets()***, dichiarata nel file di intestazione '`stdio.h`'.

La funzione ***gets()*** ottiene una stringa leggendo continuamente il contenuto della variabile '**`os.kbd.key`**'.

Listato u174.1. '`./05/lib/gets.c`'

```
#include <stdio.h>
#include <kernel/os.h>
char
*gets (char *s)
{
    int i;
    //
    // Legge os.kbd.char.
    //
    for (i = 0; i < 256; i++)
      {
        while (os.kbd.key == 0)
          {
            //
            // Attende un carattere.
            //
            ;
          }
        s[i] = os.kbd.key;
        os.kbd.key = 0;
        if (s[i] == '\n')
          {
            s[i] = 0;
            break;
          }
      }
    return s;
}
```

Realizzazione della shell

La shell è costituita dalla funzione ***shell()***, dichiarata nel file di intestazione '`app.h`', nel quale potrebbero essere inseriti i prototipi di altri tipi di applicazione, da avviare con l'aiuto della shell stessa.

Listato u174.2. '`./05/include/app/app.h`'

```
#ifndef _APP_H
#define _APP_H      1

void shell ();

#endif
```

Listato u174.3. '`./05/app/shell.c`'

```
#include <app/app.h>
#include <stdio.h>
#include <string.h>
#include <kernel/gdt.h>
#include <kernel/kernel.h>
#include <kernel/mm.h>
#include <kernel/multiboot.h>
void
shell (void)
{
    char command[256];
    //
    //
    //
    while (true)
      {
```

```
        printf ("# ");
        //
        // Legge un comando.
        //
        gets (command);
        //
        if (strcmp (command, "quit") == 0
            || strcmp (command, "q") == 0)
          {
            break;
          }
        else if (strcmp (command, "help") == 0
                 || strcmp (command, "h") == 0)
          {
            printf ("shell commands:\n");
            printf ("h|help         = this help\n");
            printf ("q|quit         = quit the shell\n");
            printf ("i mb|info mb   = "
                    "show multiboot info\n");
            printf ("i gdt|info gdt = show gdt\n");
            printf ("i mem|info mem = show memory map\n");
          }
        else if (strcmp (command, "info mb") == 0
                 || strcmp (command, "i mb") == 0)
          {
            mboot_show ();
          }
        else if (strcmp (command, "info gdt") == 0
                 || strcmp (command, "i gdt") == 0)
          {
            gdt_print (&os.gdtr);
          }
        else if (strcmp (command, "info mem") == 0
                 || strcmp (command, "i mem") == 0)
          {
            kernel_memory_show ();
            mm_list ();
          }
        else
          {
            printf ("[%s] unknown command: %s\n", __func__,
                    command);
          }
      }
}
```

La shell mostra un invito e si spetta l'inserimento di comandi molto semplici, come '**`i mem`**' per avere la mappa dell'utilizzo della memoria. Se si sbaglia non è possibile correggere e la pressione di tasti per la cancellazione provoca semplicemente la scrittura di codici non gestiti. Si osservi che anche gli spazi superflui contano come «errori».

Conclusione

Per concludere viene mostrato il listato definitivo del file '`kernel_main.c`', in cui si avvia la shell. Se con questo sistema si volesse fare qualcosa di più, basterebbe intervenire nella shell stessa, senza ritoccare ulteriormente il file '`kernel_main.c`'.

Listato u174.4. '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
#include <app/app.h>
//
// Funzione principale, da dove si avvia il kernel.
//
void
kernel_main (unsigned long magic, multiboot_t *info)
{
    //
    // Inizializza i dati relativi alla gestione dello
    // schermo VGA, quindi ripulisce lo schermo.
    //
    vga_init ();
    clear ();
    //
    // Data e orario di compilazione.
    //
    printf ("05 %s\n", BUILD_DATE);
    //
    // Cerca le informazioni «multiboot».
    //
    if (magic == 0x2BADB002)
      {
        //
        // Salva e mostra le informazioni multiboot.
        //
        mboot_info (info);
        mboot_show ();
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Predispone il timer.
        //
        timer_freq (CLOCKS_PER_SEC);
        //
        // Predispone la tastiera.
        //
        keyboard_load ();
        echo ();
        //
        // Predispone la tabella IDT.
        //
        idt();
      }
    else
      {
        printf ("[%s] no \"multiboot\" header!\n",
                __func__);
      }
    //
    // Shell.
    //
    shell ();
    //
    printf ("[%s] system halted\n", __func__);
    _Exit (0);
}
```

Nella schermata successiva si vede una breve interazione con la shell, dove appare anche un errore di digitazione.

```
# help
shell commands:
h|help         = this help
q|quit         = quit the shell
i mb|info mb   = show multiboot info
i gdt|info gdt = show gdt
i mem|info mem = show memory map
# info mb
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
# info gdt
[gdt_print] base: 0x0010E068 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001100000000
# info em...
[shell] unknown command: info em...
# info mem
[kernel_memory_show] kernel 00100000..0010E5A4  avail. 0010E5A4..01EF0000
[kernel_memory_show] text   00100000..001058F4  rodata 00105900..0010600C
[kernel_memory_show] data   0010600C..0010600C  bss    00106020..0010E5A4
[kernel_memory_show] limit  00001EF0
[mm_list] free 0010E5A8..01EF0000 size 01EEFFFC
# quit
[kernel_main] system halted
```

Caratteristiche comuni nei sistemi *BSD 1983
Partizioni nei sistemi *BSD 1983
Dispositivi tipici 1984
Riferimenti 1984

Caratteristiche comuni nei sistemi *BSD

Partizioni nei sistemi *BSD 1983
Dispositivi tipici 1984
Riferimenti 1984

A causa della sfortuna di 386BSD, che ha dovuto essere eliminato dalla distribuzione pubblica (per i motivi legali a cui si accenna nella sezione ??capitolo storia breve bsd??), lo Unix BSD «libero» si è suddiviso in tre varianti diverse: NetBSD, FreeBSD e OpenBSD.

Lo scopo di questo capitolo è quello di annotare gli elementi comuni, o comunque le caratteristiche tipiche di questi sistemi *BSD, senza entrare troppo nel dettaglio.

Partizioni nei sistemi *BSD

Anche se i sistemi *BSD riconoscono le partizioni Dos, gestiscono un sistema di partizioni autonomo, denominato *disklabel*. Quando si utilizzano architetture x86, queste partizioni speciali BSD si ottengono all'interno di una partizione Dos normale.

Per la precisione, una *disklabel* è una partizione speciale BSD, identificata da un'etichetta di riconoscimento (e questo spiega il senso del nome che gli è stato attribuito: «etichetta del disco») nella forma '`x:`'. La cosa ricorda un po' le unità a disco del Dos, con la differenza che le etichette BSD sono composte con delle lettere minuscole.

L'uso di queste etichette segue una tradizione, più o meno obbligatoria. In generale, si utilizzano solo le etichette da '`a:`' a '`h:`'; inoltre, le prime quattro hanno un significato ben preciso, come si vede nella tabella u175.1, che di solito è meglio non tentare di alterare.

Tabella u175.1. Utilizzo standard delle partizioni nei sistemi *BSD.

Etichetta	Utilizzo
a:	Partizione primaria.
b:	Partizione di scambio per la memoria virtuale.
c:	Spazio complessivo usato dal sistema *BSD.
d:	Spazio complessivo del disco.
e:, f:, g:, h:	Partizioni disponibili.

Osservando la tabella, si può notare la particolarità della partizione identificata dall'etichetta '`c:`', il cui scopo è quello di riassumere lo spazio complessivo utilizzato. A seconda della variante BSD, può darsi che l'etichetta '`c:`' svolga simultaneamente anche il ruolo della '`d:`'. Questo particolare va verificato.

L'etichetta '`b:`' è molto importante. In molti casi non c'è modo di definire una partizione di scambio differente. Anche se non dovesse essere necessaria una partizione di scambio, nel disco (o nella partizione Dos) che si sta suddividendo, conviene definire un'etichetta '`b:`' di dimensione nulla, per evitare inconvenienti spiacevoli.

In generale, se i dischi a cui si vuole accedere con il proprio sistema *BSD hanno una geometria reale diversa da quella che viene mostrata effettivamente, possono nascere degli inconvenienti. Per risolvere il problema alla radice, è sufficiente configurare la geometria in modo che corrisponda a quella reale (in mancanza di altro, si può cercare di leggere l'etichetta del disco fisso).

Storicamente, la traduzione della geometria è una tecnica nata per risolvere il problema del BIOS (il firmware degli elaboratori x86), che non era in grado di accedere a cilindri oltre il 1024-esimo. Evidentemente, il problema resta; quello che conta è che la partizione principale, corrispondente all'etichetta '`a:`', si trovi entro tale limite.

Il file di dispositivo utilizzato per identificare una partizione BSD non contiene l'informazione della partizione Dos, come avviene invece con GNU/Linux, limitandosi a specificare l'etichetta. Questo fatto ha delle conseguenze importanti: ci può essere una sola partizione BSD in un disco; per accedere alle partizioni di altri sistemi operativi, occorre creare delle etichette che vi fanno riferimento.

Dispositivi tipici

I file di dispositivo dei sistemi *BSD si trovano nella directory '`/dev/`', secondo la tradizione Unix. Purtroppo, le varianti *BSD non usano le stesse convenzioni, per cui è sempre necessario leggere la documentazione specifica per sapere quale file di dispositivo utilizzare. La tabella u175.2 elenca solo alcuni dispositivi essenziali, mostrando in particolare il nome del file corrispondente nella directory '`/dev/`'.

Tabella u175.2. Alcuni dispositivi tipici nei sistemi *BSD su architettura x86.

Dispositivi	File	I/O	IRQ	DMA	Annotazioni
Porte seriali	com0, pccom0	$3F8_{16}$	4		
	com1, pccom1	$2F8_{16}$	3		
	com2, pccom2	$3E8_{16}$	5		
Porte parallele	lpt0	378_{16}	7		
	lpt1	278_{16}	--		
	lpt2	$3BC_{16}$	--		
Dischi ATA	wd0				
	wd1				
	wd2				
	wd3				
Unità a dischetti	fd0				
	fd1				
Dischi SCSI	sd0				
	sd1				
	sd2				
	sd3				
Nastri SCSI	st0				
	st1				
CD-ROM	cd0				
	cd1				

È interessante notare il modo in cui si identificano le partizioni e le etichette relative:

```
wdnx
```

```
sdnx
```

I due modelli rappresentano un disco ATA e un disco SCSI, dove ***n*** corrisponde al disco, a partire da zero, e ***x*** corrisponde alla lettera dell'etichetta. È chiaro che manca l'indicazione della partizione Dos, per cui tutto, anche le partizioni esterne, va inteso a livello di etichetta.

Riferimenti

- Terry Lambert, Dave Burgess, *NetBSD, FreeBSD, and OpenBSD FAQ*
  http://www.cs.uu.nl/wais/html/na-dir/386bs86d-faq/part1.html
- The FreeBSD Documentation Project, *FreeBSD Handbook*
  http://www.freebsd.org/docproj/docproj.html

Minix 1987
Procurarsi il software 1987
Preparazione all'installazione 1988
Avvio 1989
Installazione 1990
Ricompilazione del kernel 1993
Parametri di avvio 1995
Configurazione della rete 1996
Personalizzazione 1997
Tastiera 1997
Altri programmi 1999
Copie di sicurezza 2000
Convivenza tra Minix e GNU/Linux 2001
Riferimenti 2001
ELKS: introduzione 2003
Strumenti di sviluppo 2003
Compilazione del kernel 2004
Immagini di dischetti già pronti 2004
Avvio di ELKS all'interno di DOSEMU 2004
Spegnimento 2005
Riferimenti 2005
ELKS: realizzazione personale 2007
File system e dischetti 2007
File di dispositivo 2007
Sistema di avvio 2008
Installazione manuale nel disco fisso 2012
Adattamento della mappa della tastiera 2013

Procurarsi il software 1987
Pacchetti essenziali 1988
Preparazione all'installazione 1988
Nomi di dispositivo riferiti alle partizioni 1988
Avvio 1989
Installazione 1990
Setup 1991
Avvio del sistema copiato nel disco fisso 1993
Installazione delle serie di dischetti 1993
Dischetto di avvio 1993
Conclusione 1993
Ricompilazione del kernel 1993
/usr/include/minix/config.h 1994
File di dispositivo 1995
Parametri di avvio 1995
Scheda di rete 1995
Configurazione della rete 1996
/etc/hosts 1996
File «/etc/hostname.file» 1996
File «/etc/resolv.conf» 1996
File «/etc/rc.net» 1996
File «/etc/rc» 1996
Personalizzazione 1997
File «/etc/passwd», «/etc/group» e «/etc/shadow» 1997
Tastiera 1997
Modifica della mappa 1998
Altri programmi 1999
Pacchetto «HTTPD.TAZ» 1999
Pacchetto «FROG.TAZ» 2000
Copie di sicurezza 2000
Archiviazione 2000
Recupero 2001
Convivenza tra Minix e GNU/Linux 2001
LILO 2001
Riferimenti 2001

Minix [1] è nato originariamente come sistema didattico. Minix non ha avuto il successo e la diffusione che avrebbe potuto avere a causa delle limitazioni della sua licenza iniziale. In seguito le cose sono cambiate, fortunatamente, perché Minix resta probabilmente l'unica possibilità reale per chi vuole utilizzare elaboratori con architettura i286 o inferiore.

Lo scopo di questo capitolo è introdurre all'uso di Minix per poter riutilizzare i vecchi elaboratori i286, in particolare, collegandoli a una rete locale TCP/IP. Le informazioni seguenti si riferiscono alla versione 2.0.0.

Procurarsi il software

Minix è ottenibile dalla rete, precisamente a partire dalla sua pagina di presentazione ufficiale, quella del suo primo autore (La sigla «AST» rappresenta le iniziali di Andrew S. Tanenbaum), oltre che dai vari siti speculari del relativo FTP.

http://www.cs.vu.nl/~ast/minix.html

Minix è nato assieme a un libro che tuttora dovrebbe essere accompagnato da un CD-ROM contenente il sistema operativo:

- Andrew S. Tanenbaum, Alber S. Woodhull, *Operating Systems: Design and Implementation*, 2/e, Prentice-Hall

Pacchetti essenziali

Minix è un sistema molto piccolo e composto da pochi pacchetti. Questi hanno alcune particolarità: i nomi sono composti con lettere maiuscole e gli archivi compressi utilizzano la combinazione '**`tar`**'+'**`compress`**' e sono evidenziati dall'uso dell'estensione '`.TAZ`'.

Per prima cosa è necessario riprodurre la coppia di dischetti '`ROOT`' e '`USR`', a partire dai file omonimi. Il primo è in grado di avviarsi e contiene un file system minimo che si installa in un disco RAM, il secondo contiene una piccola serie di programmi da innestare nella directory '`/usr/`', che servono per poter installare Minix nel disco fisso. Se si ha poca memoria a disposizione (i classici 640 Kibyte sono il minimo in assoluto per poter fare funzionare Minix), si può evitare l'utilizzo del disco RAM fondendo i due file in un solo dischetto. Data l'intenzione di questo capitolo viene descritta l'ultima di queste modalità di installazione.

Il mini sistema che si ottiene attraverso i due file appena citati, permette di installare, più o meno automaticamente, un insieme minimo di programmi contenuto nell'archivio '`USR.TAZ`'

Esistono due versioni di questi tre file: una per architettura i386 o superiore e l'altra per i microprocessori inferiori. Date le intenzioni, si devono utilizzare i file della versione denominata '**`i86`**'.

Preparazione all'installazione

Il procedimento che viene descritto è valido sia per dischetti da 1440 Kibyte che da 1200 Kibyte. I due file '`ROOT`' e '`USR`' vanno copiati uno di seguito all'altro. Utilizzando GNU/Linux, si può fare nel modo seguente:

```
# cat ROOT USR > /dev/fd0 [Invio]
```

Dal punto di vista di Minix, il dischetto inserito nella prima unità a dischetti corrisponde al dispositivo '`/dev/fd0`', come per GNU/Linux, ma risulta diviso in partizioni: l'immagine '`ROOT`' risulta essere '`/dev/fd0a`' e l'immagine '`USR`' è '`/dev/fd0c`' (la partizione '**`b`**' è vuota).[2]

L'archivio '`USR.TAZ`' deve essere suddiviso in diversi dischetti, con un procedimento un po' insolito: viene semplicemente tagliato a fettine della dimensione massima contenibile dal tipo di dischetti che si utilizza. Nel caso si tratti di dischetti da 1440 Kibyte, si può utilizzare GNU/Linux, o un altro sistema Unix, nel modo seguente:

```
# dd if=/USR.TAZ of=/dev/fd0 bs=1440k count=1 skip=0 [Invio]

# dd if=/USR.TAZ of=/dev/fd0 bs=1440k count=1 skip=1 [Invio]

# dd if=/USR.TAZ of=/dev/fd0 bs=1440k count=1 skip=2 [Invio]
```

Se si trattasse di dischetti da 1200 Kibyte, occorrerebbe modificare la dimensione del blocco, come nell'esempio seguente:

```
# dd if=/USR.TAZ of=/dev/fd0 bs=1200k count=1 skip=0 [Invio]

# dd if=/USR.TAZ of=/dev/fd0 bs=1200k count=1 skip=1 [Invio]

# dd if=/USR.TAZ of=/dev/fd0 bs=1200k count=1 skip=2 [Invio]
```

Nomi di dispositivo riferiti alle partizioni

Minix viene installato normalmente all'interno di una partizione primaria suddivisa in almeno due partizioni secondarie. La prima partizione serve a contenere il file system principale ed è di piccole dimensioni: 1440 Kibyte. La seconda serve per tutto il resto e viene innestata in corrispondenza della directory '`/usr/`'. Inizialmente le partizioni erano tre e '`/usr/`' era la terza. Attualmente, '`/usr/`' continua a essere la terza partizione e si finge che esista una seconda partizione senza alcuno spazio a disposizione.

Il primo disco fisso viene identificato dal dispositivo '`/dev/hd0`', il secondo da '`/dev/hd5`'. Le partizioni primarie del primo disco fisso vanno da '`/dev/hd1`' a '`/dev/hd4`'; quelle del secondo disco fisso da '`/dev/hd6`' a '`/dev/hd9`'. Le partizioni secondarie corrispondono al nome del dispositivo della partizione primaria con l'aggiunta di una lettera alfabetica che ne indica l'ordine.

- /dev/hd0
  - /dev/hd1
    - /dev/hd1a
    - /dev/hd1b
    - /dev/hd1c
    - ...
  - /dev/hd2
  - /dev/hd3
  - /dev/hd4
- /dev/hd5
  - /dev/hd6
  - /dev/hd7
  - /dev/hd8
  - /dev/hd9

Minix può essere installato in una partizione primaria qualunque, purché ci siano almeno 40 Mibyte a disposizione. Utilizzando la versione di Minix '**`i86`**', non conviene tentare di superare i 128 Mibyte.

Avvio

Minix utilizza un sistema di avvio piuttosto sofisticato; per fare un paragone con GNU/Linux, si tratta di qualcosa che compie le stesse funzioni di LILO, o di un cosiddetto *bootloader*.

Il sistema che svolge questa funzione in Minix si chiama *boot monitor* ed è importante capire subito come si utilizza se non si ha molta memoria RAM a disposizione, quanta ne richiederebbe un disco RAM per l'immagine '`ROOT`'.

Per cominciare, dopo aver preparato il dischetto '`ROOT`'+'`USR`', lo si inserisce senza la protezione contro la scrittura e si avvia l'elaboratore. Questo è ciò che appare.

```
Minix boot monitor 2.5

Press ESC to enter the monitor

Hit a key as follows:

    = Start Minix
```

Premendo il tasto [*Esc*] si attiva il *boot monitor*, mentre premendo [=] (si fa riferimento alla tastiera americana e questo simbolo si trova in corrispondenza della nostra lettera «ì») si avvia Minix con le impostazioni predefinite.

Dal momento che si immagina di avere a disposizione poca memoria (solo 1 Mibyte), non si può avviare Minix così, perché il contenuto dell'immagine '`ROOT`' verrebbe caricato come disco RAM. È necessario utilizzare subito il *boot monitor*.

[*Esc*]

```
[ESC]
fd0>
```

Si ottiene un invito (*prompt*), attraverso il quale possono essere utilizzati alcuni comandi importanti per predisporre l'avvio del sistema Minix. Per ottenere aiuto si può utilizzare il comando '**`help`**'.

```
fd0> help [Invio]
```

Si ottiene un riepilogo dei comandi e del modo con cui possono essere utilizzati. Le cose più importanti che si possono fare con il *boot monitor* sono: l'avvio a partire da una partizione differente da quella prestabilita; l'assegnamento o il ripristino al valore predefinito di una variabile.

Queste variabili sono solo entità riferite al sistema di avvio e la loro modifica permette di cambiare il modo con cui si avvia il kernel. Il comando '**set**' permette di elencare il contenuto di queste variabili.

```
fd0> set [Invio]

  rootdev = (ram)
  ramimagedev = (bootdev)
  ramsize = (0)
  processor = (286)
  bus = (at)
  memsize = (640)
  emssize = (330)
  video = (vga)
  chrome = (mono)
  image = (minix)
  main() = (menu)
```

I valori appaiono tutti tra parentesi tonde perché rappresentano le impostazioni predefinite. Quando si cambia qualche valore, questo appare senza le parentesi.

La prima cosa da cambiare è il dispositivo di avvio, '**rootdev**'. Si deve assegnare il nome di dispositivo riferito all'immagine 'ROOT' su dischetto. Si tratta di '/dev/fd0a', come dire, la prima partizione secondaria del dischetto. In questo caso, il nome del dispositivo può anche essere indicato senza la parte iniziale, limitandolo al solo 'fd0a'.

```
fd0> rootdev=fd0a [Invio]

fd0> set [Invio]

  rootdev = fd0a
  ramimagedev = (bootdev)
  ramsize = (0)
  processor = (286)
  bus = (at)
  memsize = (640)
  emssize = (330)
  video = (vga)
  chrome = (mono)
  image = (minix)
  main() = (menu)
```

Per avviare il sistema, basta utilizzare il comando '**boot**' senza argomenti.

```
fd0> boot [Invio]
```

In questo modo si lascia il *boot monitor* e si avvia il kernel. Una volta avviato il sistema, viene richiesto immediatamente l'innesto della seconda immagine, 'USR', contenente gli strumenti necessari all'installazione. Avendo avviato senza disco RAM, il dischetto contenente l'immagine 'ROOT' non può essere tolto e questo è il motivo per il quale deve essere contenuta nello stesso dischetto insieme a 'USR'.[3]

```
  Minix 2.0.0  Copyright 1997 Prentice-Hall, Inc.

  Executing in 16-bit protected mode

  Memory size = 970K   MINIX = 206K   RAM disk =    0K   Available = 765K

  Mon Nov  3 15:24:15 MET 1997
  Finish the name of device to mount as /usr: /dev/
```

Date le premesse, occorre specificare il nome del dispositivo corrispondente all'immagine 'USR': si tratta di 'fd0c'.

```
fd0c [Invio]

  /dev/fd0c is read-write mounted on /usr
  Starting standard daemons: update.

  Login as root and run 'setup' to install Minix.

  Minix  Release 2.0 Version 0

  noname login:
```

```
root [Invio]

#
```

Installazione

L'installazione di Minix avviene in tre fasi:

1. preparazione della partizione di destinazione;
2. trasferimento del contenuto del dischetto, ovvero delle immagini 'ROOT' e 'USR';
3. dopo il riavvio, trasferimento dell'archivio 'USR.TAZ' e possibilmente, se si dispone di una partizione di almeno 40 Mibyte, anche di 'SYS.TAZ' e 'CMD.TAZ'.

Setup

Per iniziare l'installazione, dopo aver avviato il sistema Minix dal dischetto, si utilizza lo script '**setup**'.

```
# setup [Invio]

 This is the Minix installation script.

 Note 1: If the screen blanks suddenly then hit F3 to select "software
         scrolling".

 Note 2: If things go wrong then hit DEL and start over.

 Note 3: The installation procedure is described in the manual page
         usage(8).  It will be hard without it.

 Note 4: Some questions have default answers, like this: [y]
         Simply hit RETURN (or ENTER) if you want to choose that answer.

 Note 5: If you see a colon (:) then you should hit RETURN to continue.
 :
```

[*Invio*]

Dopo la breve spiegazione, avendo premuto il tasto [*Invio*] si passa all'indicazione del tipo di tastiera. La scelta è ovvia, '**italian**', anche se non corrisponde esattamente: la barra verticale (quella per rappresentare i condotti) si trova al posto della lettera «ì» (i accentata). Durante questa fase di installazione conviene utilizzare la tastiera nazionale ('**italian**') per evitare spiacevoli incidenti quando si utilizza il programma di gestione delle partizioni.

```
 What type of keyboard do you have?  You can choose one of:

     french  italian   latin-am  scandinavn  uk      us-swap
     german  japanese  olivetti  spanish     us-std

 Keyboard type? [us-std]
```

```
italian [Invio]

 Minix needs one primary partition of at least 30 Mb (it fits in 20 Mb, but
 it needs 30 Mb if fully recompiled.  Add more space to taste.)

 If there is no free space on your disk then you have to back up one of the
 other partitions, shrink, and reinstall.  See the appropriate manuals of the
 the operating systems currently installed.  Restart your Minix installation
 after you have made space.

 To make this partition you will be put in the editor "part".  Follow the
 advice under the '!' key to make a new partition of type MINIX.  Do not
 touch an existing partition unless you know precisely what you are doing!
 Please note the name of the partition (hd1, hd2, ..., hd9, sd1, sd2, ...
 sd9) you make.  (See the devices section in usage(8) on Minix device names.)
 :
```

Il programma di Minix che permette di accedere alla tabella delle partizioni è '**part**' ed è ciò che sta per essere avviato. Come sempre, l'uso di un programma di questo genere è molto delicato: un piccolo errore mette fuori uso tutti i dati eventualmente contenuti in altre partizioni.

[*Invio*]

```
   Select device       ----first----  --geom/last--  ------sectors-----
     Device             Cyl Head Sec   Cyl Head Sec      Base      Size       Kb
     /dev/hd0                          ?   ?   ?
                        ?   ?   ?     ?   ?   ?         ?         ?         ?
 Num Sort   Type
   ?  ?    ? ?          ?   ?   ?     ?   ?   ?         ?         ?         ?
   ?  ?    ? ?          ?   ?   ?     ?   ?   ?         ?         ?         ?
   ?  ?    ? ?          ?   ?   ?     ?   ?   ?         ?         ?         ?
   ?  ?    ? ?          ?   ?   ?     ?   ?   ?         ?         ?         ?

 Type '+' or '-' to change, 'r' to read, '?' for more help, '!' for advice
```

Prima di utilizzare questo programma conviene leggere la sua guida interna, ottenibile con la pressione del tasto [*?*]. Il cursore si presenta inizialmente sull'indicazione del disco, '/dev/hd0', e può essere cambiato semplicemente premendo i tasti [+] o [-]. Una volta raggiunto il disco desiderato (in questo caso il primo disco va bene), si deve leggere la sua tabella delle partizioni, in modo da rimpiazzare tutti i punti interrogativi che riempiono lo schermo.

[*r*]

```
  Select device       ----first----  --geom/last--  ------sectors-----
    Device             Cyl Head Sec   Cyl Head Sec      Base      Size       Kb
    /dev/hd0                          615    8  17
                         0    0   0   614    7  16         0     83640    41820
Num Sort   Type
 1* hd1  86 DOS-BIG      0    1   0   613    7  16        17     83487    41743
 2  hd2  00 None         0    0   0     0    0  -1         0         0        0
 3  hd3  00 None         0    0   0     0    0  -1         0         0        0
 4  hd4  00 None         0    0   0     0    0  -1         0         0        0
```

Con questo esempio si suppone di avere solo un vecchio disco fisso MFM di circa 40 Mibyte, nel quale la prima partizione primaria risulta inizialmente dedicata al Dos. Così, basta cambiare il numero che identifica il tipo di partizione. Per farlo, vi si posiziona sopra il cursore, spostandolo con i tasti freccia, quindi si usano i tasti [+] o [-] fino a fare apparire il numero 81_{16}. Al primo intervento per cambiare un valore qualsiasi, viene richiesto esplicitamente se si intende modificare effettivamente i dati della tabella delle partizioni.

```
Do you wish to modify existing partitions (y/n) [y]
```

Una volta modificato il tipo, la prima partizione dovrebbe apparire così come segue:

```
Num Sort   Type
 1* hd1  81 MINIX        0    1   0   613    7  16        17     83487    41743
```

Quindi si conclude.

```
[q]

Save partition table?  (y/n) [y]
```

Lo script di configurazione e installazione riprende richiedendo quale sia la partizione su cui installare Minix. In questo caso si tratta della prima, cioè '`/dev/hd1`'.

```
Please finish the name of the primary partition you have created:

(Just type RETURN if you want to rerun "part")                  /dev/
```

`hd1` [*Invio*]

```
You have created a partition named:     /dev/hd1
The following subpartitions are about to be created on /dev/hd3:

       Root subpartition:      /dev/hd1a       1440 kb
       /usr subpartition:      /dev/hd1c       rest of hd1

Hit return if everything looks fine, or hit DEL to bail out if you want to
think it over.  The next step will destroy /dev/hd1.
:
```

Come accennato in precedenza, Minix viene installato in due partizioni secondarie: la prima serve a contenere il file system principale, la seconda per il resto. In seguito si possono innestare anche partizioni successive.

```
Migrating from floppy to disk...


Scanning /dev/hd1c for bad blocks.  (Hit DEL to stop the scan if are absolutely
sure that there can not be any bad blocks.  Otherwise just wait.)
```

La scansione del disco fisso è necessaria se si utilizza un vecchio disco MFM, come si suppone di avere in questo esempio, mentre può essere inutile con un disco ATA. È importante fare attenzione: se ci sono settori inutilizzabili, vengono creati alcuni file '`/usr/.Bad_*`' che **non vanno cancellati!** Alla fine, lo script procede a copiare il contenuto del dischetto nel disco fisso.

```
What is the memory size of this system in kilobytes? [4096 or more]
```

La dimensione di memoria RAM disponibile effettivamente è solo di 970 Kibyte, quindi si inserisce questo valore; se la memoria a disposizione fosse maggiore o uguale a 4 Mibyte, non occorrerebbe indicare alcunché, basterebbe solo confermare.

`970` [*Invio*]

```
Second level file system block cache set to 0 kb.
```

A questo punto, termina l'installazione del dischetto nel disco fisso e si può passare a riavviare il sistema da lì.

```
Please insert the installation ROOT floppy and type 'halt' to exit Minix.
You can type 'boot hd1' to try the newly installed Minix system.  See
"TESTING" in the usage manual.
```

```
# halt [ Invio ]

  System Halted
```


Avvio del sistema copiato nel disco fisso

Una volta conclusa l'esecuzione dello script di configurazione e installazione, si ritorna sotto il controllo del *boot monitor*, attraverso il quale è possibile avviare il sistema dalla prima partizione del disco fisso.

```
fd0> boot /dev/hd1 [ Invio ]

  Minix 2.0.0  Copyright 1997 Prentice-Hall, Inc.

  Executing in 16-bit protected mode

  at-hd0: 615x8x17

  Memory size = 970K   MINIX = 206K   RAM disk =    0K   Available = 765K

  Mon Nov  3 16:01:27 MET 1997
  Starting standard daemons: update.

  Login as root and run 'setup /usr' to install floppy sets.

  Minix  Release 2.0 Version 0

  noname login:
```

```
# root [ Invio ]
```

Il suggerimento dato all'avvio ricorda che è possibile installare altre serie di dischetti, a cominciare da '`USR.TAZ`', utilizzando il comando '**`setup /usr`**'.

Installazione delle serie di dischetti

Tra i pacchetti di Minix, '`USR.TAZ`' è essenziale e cambia a seconda del tipo di architettura (i86 o i386). Però, dal momento che c'è spazio sufficiente nel disco fisso, conviene installare anche '`SYS.TAZ`' per poter ricompilare il kernel e '`CMD.TAZ`' che contiene i sorgenti dei vari programmi di servizio.

Tutti questi pacchetti devono essere suddivisi in dischetti nel modo visto in precedenza per il caso di '`USR.TAZ`'.

```
# setup /usr [ Invio ]
```

Lo script '**`setup`**' chiede una serie di conferme.

```
  What is the size of the images on the diskettes? [all]
```

Premendo semplicemente [*Invio*] si intende che i dischetti vadano letti nella loro interezza.

```
[ Invio ]

  What floppy drive to use? [0]
```

Premendo semplicemente [*Invio*] si fa riferimento alla prima unità, '`/dev/fd0`'.

```
[ Invio ]

  Please insert input volume 1 and hit return
```

Si inserisce il primo dischetto e si conferma

```
[ Invio ]
```

Inizia la fase di estrazione di quanto contenuto nel primo dischetto, a partire dalla directory '`/usr/`'. Quando termina l'estrazione del primo dischetto, viene richiesto il successivo, fino alla conclusione.

Conviene ripetere la procedura fino a quando sono stati installati anche gli archivi '`SYS.TAZ`' e '`CMD.TAZ`'.

Dischetto di avvio

Minix è molto semplice e non è necessario un dischetto di avvio realizzato appositamente. È sufficiente il dischetto utilizzato per iniziare l'installazione. Se si hanno difficoltà con l'avviamento di Minix dal disco fisso, si può avviare il *boot monitor* dal dischetto e con quello utilizzare il comando '**`boot /dev/hd1`**'.

Conclusione

Per chiudere l'attività di Minix, si può fare nel solito modo comune a quasi tutti i sistemi Unix.

```
# shutdown -h now [ Invio ]
```

Ricompilazione del kernel

Anche Minix, nella sua semplicità, richiede una ricompilazione del kernel per la sua ottimizzazione. In particolare, per poter attivare la gestione del TCP/IP occorre passare per la configurazione e ricompilazione.

Il file del kernel, secondo la tradizione di Minix, dovrebbe trovarsi nella directory radice e avere il nome '`minix`'. Se però, invece di trattarsi di un file, si tratta di una directory, nella fase di avvio viene eseguito il file più recente contenuto in tale directory. Il kernel normale, cioè quello che si trova dopo l'installazione, dovrebbe essere '`/minix/2.0.0`'.

Per poter ricompilare il kernel occorre avere installato il pacchetto '`SYS.TAZ`'. Si procede come segue:

1. si modifica il file '`/usr/include/minix/config.h`';
2. ci si posiziona nella directory '`/usr/src/tools/`';
3. si avvia la compilazione con il comando '**`make`**'.

Al termine si ottiene il file del kernel (o immagine) corrispondente a '`/usr/src/tools/image`' che si può copiare e rinominare come si ritiene più opportuno.

/usr/include/minix/config.h

La configurazione che viene proposta deriva dagli esempi precedenti, in cui si ha una particolare penuria di memoria. Seguono solo alcuni pezzi.

```
/* If ROBUST is set to 1, writes of i-node, directory, and indirect blocks
 * from the cache happen as soon as the blocks are modified.  This gives a more
 * robust, but slower, file system.  If it is set to 0, these blocks are not
 * given any special treatment, which may cause problems if the system crashes.
 */
#define ROBUST             1    /* 0 for speed, 1 for robustness */
```

La macro '**`ROBUST`**' permette di sincronizzare le operazioni di accesso al disco. Nell'esempio mostrato si attiva questa opzione, in modo da poter utilizzare il sistema con tranquillità (e ovviamente con maggiore lentezza).

```
/* Number of slots in the process table for user processes. */
#define NR_PROCS          32
```

Il numero massimo dei processi eseguibili può essere una seria limitazione all'uso simultaneo dell'elaboratore da parte di più utenti, ma la scarsa memoria a disposizione consiglia di mantenere basso questo valore.

```
/* Enable or disable the second level file system cache on the RAM disk. */
#define ENABLE_CACHE2      0
```

Sempre a causa della carenza di memoria, è opportuno disabilitare la memoria cache.

```
/* Include or exclude device drivers.  Set to 1 to include, 0 to exclude. */
#define ENABLE_NETWORKING  1    /* enable TCP/IP code */
#define ENABLE_AT_WINI     1    /* enable AT winchester driver */
#define ENABLE_BIOS_WINI   1    /* enable BIOS winchester driver */
#define ENABLE_ESDI_WINI   1    /* enable ESDI winchester driver */
#define ENABLE_XT_WINI     0    /* enable XT winchester driver */
#define ENABLE_ADAPTEC_SCSI 0   /* enable ADAPTEC SCSI driver */
#define ENABLE_MITSUMI_CDROM 0  /* enable Mitsumi CD-ROM driver */
#define ENABLE_SB_AUDIO    0    /* enable Soundblaster audio driver */
```

In questa sezione è importante abilitare ciò che serve ed eliminare il resto. In particolare, è qui che si attiva la connettività TCP/IP, che non risulta attivata in modo predefinito.

```
/* NR_CONS, NR_RS_LINES, and NR_PTYS determine the number of terminals the
 * system can handle.
 */
#define NR_CONS            2    /* # system consoles (1 to 8) */
#define NR_RS_LINES        1    /* # rs232 terminals (0, 1, or 2) */
#define NR_PTYS            2    /* # pseudo terminals (0 to 64) */
```

Il numero predefinito di console virtuali è due, ma può essere espanso, sempre che ciò possa avere senso date le limitazioni del sistema. Invece è importante attivare gli pseudoterminali, cioè il numero massimo di connessioni remote. Volendo gestire la rete, è il caso di indicare almeno uno pseduoterminale.

Per modificare il file '`/usr/include/minix/config.h`' si può utilizzare '**`vi`**', che è un collegamento a '**`elvis`**', oppure '**`elle`**'.[4]

Si procede con la compilazione.


```
# cd /usr/src/tools [Invio]
```

```
# make [Invio]
```

Al termine della compilazione, se non sono occorsi incidenti, si ottiene il file '`image`'.

```
# cp image /minix/rete.0.1 [Invio]
```

Questo dovrebbe bastare, trattandosi del file più recente nella directory '`/minix/`', è anche quello che viene avviato la volta successiva.

```
# shutdown -h [Invio]
```

File di dispositivo

Quando si ricompila il kernel è probabile che si renda necessaria la creazione di file di dispositivo che in altre situazioni non sarebbero necessari. Nel caso della gestione della rete, sono necessari i file seguenti:

- '`/dev/eth`';
- '`/dev/ip`';
- '`/dev/tcp`';
- '`/dev/udp`';
- '`/dev/ttyp0`' e successivi;
- '`/dev/ptyp0`' e successivi.

Questo ragionamento vale anche per le console virtuali: se si vogliono molte console, forse è necessario aggiungere i file relativi.

Probabilmente c'è già tutto ciò di cui si può avere bisogno, ma se manca si può creare con lo script '**`MAKEDEV`**'.

```
MAKEDEV dispositivo
```

Per esempio, trovandosi già nella directory '`/dev/`', si può creare il dispositivo '`/dev/tcp`' nel modo seguente:

```
# MAKEDEV tcp [Invio]
```

Parametri di avvio

Anche Minix richiede alcuni parametri di avvio in presenza di hardware particolare. La gestione di questi avviene in modo molto semplice attraverso il *boot monitor*: basta definire una nuova variabile, assegnandole il valore corretto.

Scheda di rete

Per gestire una rete occorre una scheda di rete Ethernet. Nell'esempio seguente si immagina di disporre di una scheda compatibile con il modello NE2000 configurata con indirizzo di I/O 300_{16} e IRQ 11.

Il parametro di avvio per ottenere il riconoscimento della scheda Ethernet è '**`DPETH`*`n`***', dove ***`n`*** è il numero della scheda, a partire da zero.

```
DPETHn=indirizzo_i/o : irq : indirizzo_di_memoria
```

La scheda NE2000 non utilizza alcun indirizzo di memoria, quindi, per il nostro esempio occorre il parametro seguente:

```
DPETH0=300:11
```

Come si vede, l'indirizzo di I/O è espresso implicitamente in esadecimale e l'IRQ in decimale, mentre l'indirizzo di memoria viene omesso trattandosi di una NE2000. Per inserire tale parametro si utilizza il *boot monitor* nel modo seguente:

```
hd0> DPETH0=300:11 [Invio]
```

```
hd0> save [Invio]
```

L'ultima istruzione, '**`save`**', salva questo parametro che altrimenti dovrebbe essere indicato ogni volta che si avvia il sistema.

Se la scheda di rete viene riconosciuta, all'avvio appare il messaggio seguente:

```
Minix 2.0.0  Copyright 1997 Prentice-Hall, Inc.

Executing in 16-bit protected mode

ne2000: NE2000 at 300:11
```

Configurazione della rete

La configurazione della rete va fatta con cura, in modo da non avere bisogno di alcuni demoni che permettono una sorta di autoconfigurazione. Negli esempi seguenti si configura il nuovo sistema Minix tenendo conto di questa situazione:

- *dinkel.brot.dg*, IP 192.168.1.1, servizio DNS e router predefinito;
- *minix.brot.dg*, IP 192.168.1.25, elaboratore Minix.

Per quanto possibile, si fa in modo di non avere bisogno del DNS.

/etc/hosts

Volendo attivare localmente la risoluzione dei nomi e degli indirizzi è necessario il file '`/etc/hosts`', che va configurato come al solito, esattamente come si fa con GNU/Linux.

```
127.0.0.1       localhost
192.168.1.1     dinkel.brot.dg
192.168.1.25    minix.brot.dg
```

File «/etc/hostname.file»

Il file '`/etc/hostname.file`' serve solo a definire il nome dell'elaboratore locale, in senso generale. Non ha niente a che vedere con le interfacce di rete.

```
# echo "minix.brot.dg" > /etc/hostname.file [Invio]
```

File «/etc/resolv.conf»

Il file '`/etc/resolv.conf`' permette di indicare gli indirizzi dei nodi che forniscono un servizio DNS. Nell'esempio proposto si vuole fare in modo che il sistema di risoluzione dei nomi avvenga localmente, per mezzo di quanto contenuto nel file '`/etc/hosts`'. Per questo viene indicato come servente DNS anche l'indirizzo locale (*loopback*).

```
nameserver 127.0.0.1
nameserver 192.168.1.1
```

File «/etc/rc.net»

Lo script '`/etc/rc.net`' viene utilizzato da '`/etc/rc`' per attivare la rete. Lo si può utilizzare per attivare l'interfaccia di rete e per definire l'instradamento verso il router (l'instradamento verso la rete connessa all'interfaccia è predefinito).

```
# Attiva l'interfaccia e l'instradamento verso la sua rete.
ifconfig -h 192.168.1.25

# Definisce l'instradamento predefinito verso il router
add_route -g 192.168.1.1
```

File «/etc/rc»

Probabilmente, è utile ritoccare il file '`/etc/rc`', per eliminare l'avvio automatico di alcuni demoni inutili dal momento che la rete è configurata. Quello che segue è il pezzo che attiva la gestione della rete.

```
# Network initialization.
(</dev/eth </dev/tcp) 2>/dev/null && net=true   # Is there a TCP/IP server?

if [ "$net" -a -f /etc/rc.net ]
then
        # There is a customized TCP/IP initialization script; run it.
        . /etc/rc.net
elif [ "$net" ] && [ "'hostaddr -e'" = 0:0:0:0:0:0 ]
then
        # No network hardware, configure a fixed address to run TCP/IP alone.
        ifconfig -h 192.9.200.1
fi

if [ "$net" ]
then
        echo -n "Starting network daemons: "
        for daemon in rarpd nonamed irdpd talkd
        do
                if [ -f /usr/bin/$daemon ]
                then
                        echo -n " $daemon"
                        $daemon &
                fi
        done
        echo .

        # Get the nodename from the DNS and set it.
        hostaddr -a >/etc/hostname.file || echo noname >/etc/hostname.file

        echo -n "Starting network services:"
        for pair in 'shell in.rshd' 'login in.rld' \
                        'telnet in.telnetd' 'ftp in.ftpd'
        do
                set $pair
                if [ -f /usr/bin/$2 ]
                then
                        echo -n " $1"
                        tcpd $1 /usr/bin/$2 &
                fi
        done
        echo .
fi
```

Vale la pena di modificare quanto segue:

```
if [ "$net" ]
then
        echo -n "Starting network daemons: "
        for daemon in nonamed talkd ### rarpd nonamed irdpd talkd
        do
        ...
```

Nel pezzo precedente non vengono avviati i demoni '**`rarpd`**' e '**`irdpd`**', che sono necessari rispettivamente per ottenere l'indirizzo IP in base all'indirizzo hardware della scheda Ethernet e a definire gli instradamenti verso i router. Eventualmente, si potrebbe anche evitare di avviare '**`talkd`**' se non si intende utilizzare '**`talk`**'. Il demone '**`nonamed`**' è necessario se non si vuole essere obbligati ad avere un servizio DNS esterno; in pratica è necessario perché venga interpretato il contenuto del file '`/etc/hosts`'.

Personalizzazione

Il sistema risulta configurato in maniera piuttosto disordinata, a cominciare dal fatto che la directory personale dell'utente '**`root`**' corrisponde alla directory radice; così, al suo interno si trovano i file di configurazione dell'amministratore. Probabilmente, la prima cosa da fare è quella di creare una directory '`/root/`', porvi al suo interno i file di configurazione (dovrebbe trattarsi di '`.ellepro.b1`', '`.exrc`' e '`.profile`'), modificando anche il file '`/etc/passwd`' in modo da assegnare all'utente '**`root`**' questa nuova directory.

File «/etc/passwd», «/etc/group» e «/etc/shadow»

Minix, nonostante la sua semplicità, utilizza le parole d'ordine oscurate (*shadow password*). Pertanto, se si tenta di inserire un utente manualmente, occorre intervenire anche su questo file, '`/etc/shadow`', altrimenti l'utente non può accedere.

Il file '`/etc/group`', se non va bene com'è, deve essere modificato manualmente, mentre per gli utenti conviene affidarsi allo script '**`adduser`**'.

```
adduser utente gruppo directory_home
```

Dopo aver creato un utente, come al solito è opportuno utilizzare il programma '**`passwd`**' per assegnare la parola d'ordine.[5]

Tastiera

La mappa della tastiera viene definita attraverso il programma '**`loadkeys`**' e il file contenente la mappa desiderata. Per cui,

```
# loadkeys ./tastiera.map [Invio]
```

permette di caricare la mappa del file '`tastiera.map`' contenuto nella directory corrente.

La mappa della tastiera, secondo la scelta fatta durante l'installazione di Minix, avviene per mezzo del file '`/etc/keymap`': se lo

script ‘/etc/rc’ lo trova durante la fase di avvio, lo carica attraverso ‘**loadkeys**’.

Modifica della mappa

La configurazione della tastiera italiana, per quanto riguarda la versione 2.0 di Minix, non è perfetta. Per modificare la mappa occorre intervenire sul file ‘/usr/src/kernel/keymaps/italian.src’. Dopo la modifica si deve compilare il sorgente in modo da ottenere il file ‘/usr/src/kernel/keymaps/italian.map’. Al termine, questo file va copiato e rinominato in modo da sostituire ‘/etc/keymap’.

Il sorgente corretto potrebbe apparire come nell’esempio seguente, in particolare, per ottenere la tilde (‘**~**’) si deve usare la combinazione [*AltGr ì*], mentre per ottenere l’apostrofo inverso (‘**`**’) si deve usare la combinazione [*AltGr ’*]. I caratteri che si trovano oltre il settimo bit, vengono rappresentati in ottale.

```
/* Modified by Daniele Giacomini  daniele @ swlibero.org    1998.12.22  */
/* Keymap for Italian standard keyboard, similar to Linux layout.       */

u16_t keymap[NR_SCAN_CODES * MAP_COLS] = {

/* scan-code            !Shift  Shift   Alt     AltGr   Alt+Sh  Ctrl    */
/* ========================================================================= */
/* 00 - none    */      0,      0,      0,      0,      0,      0,
/* 01 - ESC     */      C('['), C('['), CA('['),C('['), C('['), C('['),
/* 02 - '1'     */      '1',    '!',    A('1'), '1',    '!',    C('A'),
/* 03 - '2'     */      '2',    '"',    A('2'), '2',    '@',    C('@'),
/* 04 - '3'     */      '3',    0234,   A('3'), '3',    0234,   C('C'),
/* 05 - '4'     */      '4',    '$',    A('4'), '4',    '$',    C('D'),
/* 06 - '5'     */      '5',    '%',    A('5'), '5',    '%',    C('E'),
/* 07 - '6'     */      '6',    '&',    A('6'), '6',    '&',    C('F'),
/* 08 - '7'     */      '7',    '/',    A('7'), '{',    '/',    C('G'),
/* 09 - '8'     */      '8',    '(',    A('8'), '[',    '(',    C('H'),
/* 10 - '9'     */      '9',    ')',    A('9'), ']',    ')',    C('I'),
/* 11 - '0'     */      '0',    '=',    A('0'), '}',    '=',    C('@'),
/* 12 - '-'     */      '\'',   '?',    A('\''),'\'',   '?',    C('@'),
/* 13 - '='     */      0215,   '^',    0215,   '~',    '^',    C('^'),
/* 14 - BS      */      C('H'), C('H'), CA('H'),C('H'), C('H'), 0177,
/* 15 - TAB     */      C('I'), C('I'), CA('I'),C('I'), C('I'), C('I'),
/* 16 - 'q'     */      L('q'), 'Q',    A('q'), 'q',    'Q',    C('Q'),
/* 17 - 'w'     */      L('w'), 'W',    A('w'), 'w',    'W',    C('W'),
/* 18 - 'e'     */      L('e'), 'E',    A('e'), 'e',    'E',    C('E'),
/* 19 - 'r'     */      L('r'), 'R',    A('r'), 'r',    'R',    C('R'),
/* 20 - 't'     */      L('t'), 'T',    A('t'), 't',    'T',    C('T'),
/* 21 - 'y'     */      L('y'), 'Y',    A('y'), 'y',    'Y',    C('Y'),
/* 22 - 'u'     */      L('u'), 'U',    A('u'), 'u',    'U',    C('U'),
/* 23 - 'i'     */      L('i'), 'I',    A('i'), 'i',    'I',    C('I'),
/* 24 - 'o'     */      L('o'), 'O',    A('o'), 'o',    'O',    C('O'),
/* 25 - 'p'     */      L('p'), 'P',    A('p'), 'p',    'P',    C('P'),
/* 26 - '['     */      0212,   0202,   0212,   '[',    '{',    C('['),
/* 27 - ']'     */      '+',    '*',    A('+'), ']',    '}',    C(']'),
/* 28 - CR/LF   */      C('M'), C('M'), CA('M'),C('M'), C('M'), C('J'),
/* 29 - Ctrl    */      CTRL,   CTRL,   CTRL,   CTRL,   CTRL,   CTRL,
/* 30 - 'a'     */      L('a'), 'A',    A('a'), 'a',    'A',    C('A'),
/* 31 - 's'     */      L('s'), 'S',    A('s'), 's',    'S',    C('S'),
/* 32 - 'd'     */      L('d'), 'D',    A('d'), 'd',    'D',    C('D'),
/* 33 - 'f'     */      L('f'), 'F',    A('f'), 'f',    'F',    C('F'),
/* 34 - 'g'     */      L('g'), 'G',    A('g'), 'g',    'G',    C('G'),
/* 35 - 'h'     */      L('h'), 'H',    A('h'), 'h',    'H',    C('H'),
/* 36 - 'j'     */      L('j'), 'J',    A('j'), 'j',    'J',    C('J'),
/* 37 - 'k'     */      L('k'), 'K',    A('k'), 'k',    'K',    C('K'),
/* 38 - 'l'     */      L('l'), 'L',    A('l'), 'l',    'L',    C('L'),
/* 39 - ';'     */      0225,   0207,   0225,   '@',    '@',    C('@'),
/* 40 - '\''    */      0205,   0370,   0205,   '#',    '#',    C('@'),
/* 41 - '`'     */      '\\',   '|',    '\\',   '\\',   '|',    C('\\'),
/* 42 - l. SHIFT*/      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 43 - '\\'    */      0227,   '|',    0227,   0227,   '|',    C('@'),
/* 44 - 'z'     */      L('z'), 'Z',    A('z'), 'z',    'Z',    C('Z'),
/* 45 - 'x'     */      L('x'), 'X',    A('x'), 'x',    'X',    C('X'),
/* 46 - 'c'     */      L('c'), 'C',    A('c'), 'c',    'C',    C('C'),
/* 47 - 'v'     */      L('v'), 'V',    A('v'), 'v',    'V',    C('V'),
/* 48 - 'b'     */      L('b'), 'B',    A('b'), 'b',    'B',    C('B'),
/* 49 - 'n'     */      L('n'), 'N',    A('n'), 'n',    'N',    C('N'),
/* 50 - 'm'     */      L('m'), 'M',    A('m'), 'm',    'M',    C('M'),
/* 51 - ','     */      ',',    ';',    A(','), ',',    ';',    C('@'),
/* 52 - '.'     */      '.',    ':',    A('.'), '.',    ':',    C('@'),
/* 53 - '/'     */      '-',    '_',    A('-'), '-',    '_',    C('_'),
/* 54 - r. SHIFT*/      SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 55 - '*'     */      '*',    '*',    A('*'), '*',    '*',    C('M'),
/* 56 - ALT     */      ALT,    ALT,    ALT,    ALT,    ALT,    ALT,
/* 57 - ' '     */      ' ',    ' ',    A(' '), ' ',    ' ',    C('@'),
/* 58 - CapsLck */      CALOCK, CALOCK, CALOCK, CALOCK, CALOCK, CALOCK,
/* 59 - F1      */      F1,     SF1,    AF1,    AF1,    ASF1,   CF1,
/* 60 - F2      */      F2,     SF2,    AF2,    AF2,    ASF2,   CF2,
/* 61 - F3      */      F3,     SF3,    AF3,    AF3,    ASF3,   CF3,
/* 62 - F4      */      F4,     SF4,    AF4,    AF4,    ASF4,   CF4,
/* 63 - F5      */      F5,     SF5,    AF5,    AF5,    ASF5,   CF5,
/* 64 - F6      */      F6,     SF6,    AF6,    AF6,    ASF6,   CF6,
/* 65 - F7      */      F7,     SF7,    AF7,    AF7,    ASF7,   CF7,
/* 66 - F8      */      F8,     SF8,    AF8,    AF8,    ASF8,   CF8,
/* 67 - F9      */      F9,     SF9,    AF9,    AF9,    ASF9,   CF9,
/* 68 - F10     */      F10,    SF10,   AF10,   AF10,   ASF10,  CF10,
/* 69 - NumLock */      NLOCK,  NLOCK,  NLOCK,  NLOCK,  NLOCK,  NLOCK,
/* 70 - ScrLock */      SLOCK,  SLOCK,  SLOCK,  SLOCK,  SLOCK,  SLOCK,
/* 71 - Home    */      HOME,   '7',    AHOME,  AHOME,  '7',    CHOME,
/* 72 - CurUp   */      UP,     '8',    AUP,    AUP,    '8',    CUP,
/* 73 - PgUp    */      PGUP,   '9',    APGUP,  APGUP,  '9',    CPGUP,
/* 74 - '-'     */      NMIN,   '-',    ANMIN,  ANMIN,  '-',    CNMIN,
/* 75 - Left    */      LEFT,   '4',    ALEFT,  ALEFT,  '4',    CLEFT,
/* 76 - MID     */      MID,    '5',    AMID,   AMID,   '5',    CMID,
/* 77 - Right   */      RIGHT,  '6',    ARIGHT, ARIGHT, '6',    CRIGHT,
/* 78 - '+'     */      PLUS,   '+',    APLUS,  APLUS,  '+',    CPLUS,
/* 79 - End     */      END,    '1',    AEND,   AEND,   '1',    CEND,
/* 80 - Down    */      DOWN,   '2',    ADOWN,  ADOWN,  '2',    CDOWN,
/* 81 - PgDown  */      PGDN,   '3',    APGDN,  APGDN,  '3',    CPGDN,
/* 82 - Insert  */      INSRT,  '0',    AINSRT, AINSRT, '0',    CINSRT,
/* 83 - Delete  */      0177,   '.',    A(0177),0177,   '.',    0177,
/* 84 - Enter   */      C('M'), C('M'), CA('M'),C('M'), C('M'), C('J'),
/* 85 - ???     */      0,      0,      0,      0,      0,      0,
/* 86 - ???     */      '<',    '>',    A('<'), '|',    '>',    C('@'),
/* 87 - F11     */      F11,    SF11,   AF11,   AF11,   ASF11,  CF11,
/* 88 - F12     */      F12,    SF12,   AF12,   AF12,   ASF12,  CF12,
/* 89 - ???     */      0,      0,      0,      0,      0,      0,
/* 90 - ???     */      0,      0,      0,      0,      0,      0,
/* 91 - ???     */      0,      0,      0,      0,      0,      0,
/* 92 - ???     */      0,      0,      0,      0,      0,      0,
/* 93 - ???     */      0,      0,      0,      0,      0,      0,
/* 94 - ???     */      0,      0,      0,      0,      0,      0,
/* 95 - ???     */      0,      0,      0,      0,      0,      0,
/* 96 - EXT_KEY */      EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY, EXTKEY,
/* 97 - ???     */      0,      0,      0,      0,      0,      0,
/* 98 - ???     */      0,      0,      0,      0,      0,      0,
/* 99 - ???     */      0,      0,      0,      0,      0,      0,
/*100 - ???     */      0,      0,      0,      0,      0,      0,
/*101 - ???     */      0,      0,      0,      0,      0,      0,
/*102 - ???     */      0,      0,      0,      0,      0,      0,
/*103 - ???     */      0,      0,      0,      0,      0,      0,
/*104 - ???     */      0,      0,      0,      0,      0,      0,
/*105 - ???     */      0,      0,      0,      0,      0,      0,
/*106 - ???     */      0,      0,      0,      0,      0,      0,
/*107 - ???     */      0,      0,      0,      0,      0,      0,
/*108 - ???     */      0,      0,      0,      0,      0,      0,
/*109 - ???     */      0,      0,      0,      0,      0,      0,
/*110 - ???     */      0,      0,      0,      0,      0,      0,
/*111 - ???     */      0,      0,      0,      0,      0,      0,
/*112 - ???     */      0,      0,      0,      0,      0,      0,
/*113 - ???     */      0,      0,      0,      0,      0,      0,
/*114 - ???     */      0,      0,      0,      0,      0,      0,
/*115 - ???     */      0,      0,      0,      0,      0,      0,
/*116 - ???     */      0,      0,      0,      0,      0,      0,
/*117 - ???     */      0,      0,      0,      0,      0,      0,
/*118 - ???     */      0,      0,      0,      0,      0,      0,
/*119 - ???     */      0,      0,      0,      0,      0,      0,
/*120 - ???     */      0,      0,      0,      0,      0,      0,
/*121 - ???     */      0,      0,      0,      0,      0,      0,
/*122 - ???     */      0,      0,      0,      0,      0,      0,
/*123 - ???     */      0,      0,      0,      0,      0,      0,
/*124 - ???     */      0,      0,      0,      0,      0,      0,
/*125 - ???     */      0,      0,      0,      0,      0,      0,
/*126 - ???     */      0,      0,      0,      0,      0,      0,
/*127 - ???     */      0,      0,      0,      0,      0,      0
};
```


Dopo la modifica, si avvia la compilazione.

```
# cd /usr/src/kernel/keymaps/ [Invio]

# make [Invio]
```

Vengono generati tutti i file di configurazione che non siano già presenti (se si vuole ripetere la compilazione occorre prima rimuovere il file ‘italian.map’).

```
# cp italian.map /etc/keymap [Invio]
```

Al prossimo riavvio si utilizza così la nuova mappa.

Altri programmi

Il software a disposizione per Minix non è molto e può essere trovato negli stessi FTP da cui si accede ai file del sistema operativo citati qui. In particolare, tra i programmi riferiti alla rete, vale la pena di ricordare i pacchetti ‘HTTPD.TAZ’ e ‘FROG.TAZ’. Il primo è un servente HTTP molto semplice e il secondo è un programma per il tracciamento dell’instradamento (simile a Traceroute). Sono entrambi molto utili e compilabili facilmente.

Pacchetto «HTTPD.TAZ»

Minix dispone di un servente HTTP elementare e lo si trova distribuito nel pacchetto ‘HTTPD.TAZ’. I pacchetti supplementari, come questo, vanno installati a partire dalla directory ‘/usr/local/’ e i sorgenti vanno collocati in ‘/usr/local/src/’.

```
# cd /usr/local/src [Invio]
```

Supponendo che il file ‘HTTPD.TAZ’ si trovi nel dischetto, innestato nella directory ‘/mnt/’, si può agire come segue:

```
# cat /mnt/HTTPD.TAZ | compress -d | tar xvf -
```
[*Invio*]

Si ottiene la creazione della directory ‘`httpd/`’ a partire dalla posizione corrente, cioè ‘`/usr/local/src/`’.

```
# cd httpd
```
[*Invio*]

Si procede con la compilazione.

```
# make
```
[*Invio*]

Quindi si installa il programma.

```
# make install
```
[*Invio*]

L’installazione non si occupa di copiare i file di configurazione: bisogna farlo manualmente.

```
# cp httpd.conf /etc
```
[*Invio*]

```
# cp httpd.mtype /etc
```
[*Invio*]

Il demone ‘**`httpd`**’, installato in ‘`/usr/local/bin/`’, ha bisogno di un utente ‘**`www`**’, in base alla configurazione predefinita del file ‘`/etc/httpd.conf`’ che non serve modificare.

```
# adduser www operator /usr/home/www
```
[*Invio*]

Naturalmente la scelta della directory personale di questo utente fittizio è solo un fatto di gusto personale. Sempre in base alla configurazione predefinita, occorre aggiungere alla directory personale la directory ‘`exec/`’ e all’interno di questa si devono collocare un paio di file.

```
# mkdir /usr/home/www/exec
```
[*Invio*]

```
# cp dir2html /usr/home/www/exec
```
[*Invio*]

```
# cp dir2html.sh /usr/home/www/exec
```
[*Invio*]

Per ultimo, occorre avviare il demone. Per questo conviene ritoccare il file ‘`/etc/rc.net`’ in modo da aggiungere la riga seguente:

```
tcpd   http   /usr/local/bin/httpd &
```

Pacchetto «FROG.TAZ»

«

‘**`frog`**’ è un programma per il tracciamento dell’instradamento. È semplice, ma anche molto importante. L’estrazione dell’archivio avviene nel modo solito, così come la compilazione e l’installazione.

```
# cd /usr/local/src
```
[*Invio*]

```
# cat /mnt/FROG.TAZ | compress -d | tar xvf -
```
[*Invio*]

```
# cd frog
```
[*Invio*]

```
# make
```
[*Invio*]

```
# make install
```
[*Invio*]

Tutto qui.

Copie di sicurezza

«

Le copie di sicurezza possono essere fatte soltanto utilizzano ‘**`tar`**’ e ‘**`compress`**’. Dal momento che il sistema è organizzato in modo piuttosto rigido, con una partizione principale molto piccola e una partizione ‘`/usr/`’, normalmente conviene preoccuparsi solo di questa seconda partizione. Per la prima converrebbe realizzare un dischetto di avvio e installazione con gli stessi file di configurazione, compresi ‘`/etc/passwd`’, ‘`/etc/group`’ e ‘`/etc/shadow`’.

Archiviazione

«

Se si dispone di abbastanza spazio libero nella partizione ‘`/usr/`’, se ne può fare la copia di sicurezza in un file collocato all’interno della stessa partizione. Successivamente si può scaricare su dischetti. Si può procedere nel modo seguente:

```
# cd /usr
```
[*Invio*]

```
# tar cf - .  | compress -c > .BKP.TAZ
```
[*Invio*]

Si ottiene il file ‘`/usr/.BKP.TAZ`’ contenente la copia di quanto contenuto nella directory corrente ‘`/usr/`’. Successivamente si può copiare il file ottenuto, a pezzi, su una serie di dischetti inizializzati in precedenza. Si comincia con l’inizializzazione e si suppone di disporre di dischetti da 1440 Kibyte.

```
# format /dev/fd0 1440
```
[*Invio*]

...

Una volta preparati i dischetti, si può scaricare il file nei dischetti.

```
# dd if=/usr/.BKP.TAZ of=/dev/fd0 bs=1440k count=1 skip=0
```
[*Invio*]

```
# dd if=/usr/.BKP.TAZ of=/dev/fd0 bs=1440k count=1 skip=2
```
[*Invio*]

...

Recupero

«

Per recuperare un sistema archiviato nel modo mostrato nella sezione precedente, si deve cominciare dall’installazione con il dischetto iniziale. La cosa migliore sarebbe l’utilizzo di un dischetto modificato opportunamente in modo che i file di configurazione corrispondano a quanto utilizzato nel proprio sistema.

Dopo l’installazione iniziale che consiste nel trasferimento di quanto contenuto nel dischetto iniziale nel disco fisso, si procede con l’installazione della copia (preparata in precedenza) della partizione collocata a partire da ‘`/usr/`’.

```
# setup /usr
```
[*Invio*]

Uno dopo l’altro vengono richiesti tutti i dischetti.

Convivenza tra Minix e GNU/Linux

«

Se lo si desidera, si può fare convivere Minix assieme a GNU/Linux, nello stesso disco fisso, su partizioni distinte. Ma installare Minix in una partizione libera di un disco in cui GNU/Linux è già stato installato richiede prudenza e attenzione.

- L’installazione di Minix provoca l’alterazione dell’MBR del disco fisso, di conseguenza, al termine si avvia solo Minix. Quindi, prima di installare Minix occorre preparare uno o più dischi di avvio di GNU/Linux, in modo da poter in seguito ripristinare il sistema di avvio attraverso LILO.
- Quando si conclude il lavoro con Minix e si esegue un riavvio con un semplice [*Ctrl Alt Canc*], si ottiene un avvio a caldo (*warm boot*), ma se dopo si vuole avviare un kernel Linux, **questo non può essere caricato**. Pertanto, è necessario un riavvio a freddo, al limite attraverso lo spegnimento e la riaccensione dell’elaboratore.

LILO

«

Una volta che si è riusciti a fare riavviare il sistema GNU/Linux, con i dischetti di avvio a cui si faceva riferimento in precedenza, conviene modificare il file ‘`/etc/lilo.conf`’ in modo che si possa scegliere tra l’avvio di GNU/Linux, Minix ed eventualmente altro.

Supponendo di avere installato Minix nella seconda partizione del primo disco fisso, le righe necessarie nel file ‘`/etc/lilo.conf`’ sono quelle seguenti:

```
other=/dev/hda2
        label=minix
        table=/dev/hda
```

Volendo supporre che Minix sia stato installato nel secondo disco fisso, sempre nella seconda partizione, le righe sarebbero quelle seguenti:

```
other=/dev/hdb2
        label=minix
        table=/dev/hdb
        loader=/boot/chain.b
```

In pratica, è esattamente ciò che si fa quando si vuole controllare l’avvio del Dos.

Riferimenti

Minix è un sistema operativo molto limitato rispetto a GNU/Linux. Resta comunque l'unica opportunità, almeno per ora, di fronte a vecchi elaboratori i286 o inferiori.

Molti particolari importanti non sono stati descritti, ma le informazioni relative sono comunque accessibili dai siti FTP di distribuzione di Minix e dalla documentazione interna costituita delle pagine di manuale ('**man**').

- Andrew S. Tanenbaum, Alber S. Woodhull, *Operating Systems: Design and Implementation*, 2/e, Prentice-Hall
- *http://www.cs.vu.nl/~ast/minix.html*

[1] **Minix** licenza simile a BSD

[2] Se si trattasse della seconda unità a dischetti, si parlerebbe di '`/dev/fd1`', '`/dev/fd1a`' e '`/dev/fd1c`'.

[3] Una cosa da sapere subito è che Minix non utilizza la sequenza [*Ctrl C*] per interrompere un programma. Per questo si usa il tasto [*Canc*] da solo.

[4] Il programma '**elvis**' in particolare, è molto tradizionale: i tasti freccia generano proprio le lettere corrispondenti e quindi non possono essere usati durante la fase di inserimento. '**elle**' è un Emacs ridotto al minimo.

[5] Lo script '**adduser**' si avvale della directory personale dell'utente '**ast**' per inserire i file di configurazione iniziali. Questa directory, corrispondente a '`/usr/ast/`', svolge il ruolo di scheletro delle directory personali da creare. Volendo si può realizzare un proprio script per rendere la cosa più elegante.

ELKS: introduzione

Strumenti di sviluppo 2003
Compilazione del kernel 2004
Immagini di dischetti già pronti 2004
Avvio di ELKS all'interno di DOSEMU 2004
Spegnimento 2005
Riferimenti 2005

ELKS, [1] ovvero *Embeddable linux kernel subset*, è un sistema operativo estremamente ridotto, in grado di funzionare con microprocessori i86 (16 bit), a partire dai primi (i8086 e i8088). Come dichiara il nome, si tratta di un sistema dove il kernel è derivato da Linux.

Le funzionalità disponibili sono minime e difficilmente il suo sviluppo può arrivare a un buon livello di affidabilità. Inoltre, sono sempre di meno i programmatori competenti e interessati a questo tipo di piattaforma; tuttavia, si tratta di un lavoro che potrebbe essere utile a livello didattico e sarebbe un vero peccato che venisse abbandonato del tutto.

Strumenti di sviluppo

Per compilare il kernel e i programmi di servizio che compongono il sistema operativo, è necessario un compilatore apposito, che però si usa su un sistema GNU/Linux standard. Il pacchetto del compilatore e degli strumenti di sviluppo associati è denominato Dev86 ed è distribuito normalmente in forma sorgente, assieme ai sorgenti di ELKS.

Una volta scaricato il pacchetto Dev86, questo può essere espanso in una directory qualunque nell'elaboratore GNU/Linux, come mostrato dall'esempio seguente, dove però, successivamente possa acquisire i privilegi dell'utente '**root**':

```
$ cd [Invio]

$ mkdir ELKS [Invio]

$ cd ELKS [Invio]

# tar xzvf Dev86src-0.16.0.tar.gz [Invio]
```

Si ottiene la directory '`dev86-0.16.0/`' che si articola ulteriormente. Terminata l'installazione occorre compilare questi sorgenti e installarli. In questo caso si prevede di installare Dev86 a partire da '`/opt/dev86/`':

```
$ cd dev86-0.16.0 [Invio]

$ make PREFIX=/opt/dev86/ [Invio]
```

Viene richiesto di intervenire su alcuni indicatori (*flag*); in generale dovrebbe andare bene ciò che viene proposto in modo predefinito:

```
  1) (ON)    Library of bcc helper functions
  2) (ON)    Minimal syscalls for BIOS level
  3) (ON)    Unix error functions
  4) (ON)    Management for /etc/passwd /etc/group /etc/utmp
  5) (OFF)   Linux-i386 system call routines GCC
  6) (ON)    GNU termcap routines
  7) (ON)    Bcc 386 floating point
  8) (ON)    Linux-i386 system call routines
  9) (ON)    Example kernel include files and syscall.dat
 10) (ON)    Malloc routines
 11) (ON)    Various unix lib functions
 12) (ON)    Msdos system calls
 13) (ON)    Regular expression lib
 14) (ON)    Stdio package
 15) (ON)    String and memory manipulation
 16) (ON)    Linux-8086 system call routines
 17) (ON)    Termios functions
 18) (ON)    Unix time manipulation functions.

Select config option to flip [or quit] > quit [Invio]
```

Al termine della compilazione si passa all'installazione, cominciando dalla creazione della directory '`/opt/dev86/`'. Per fare questo occorrono i privilegi dell'utente '**root**':

```
$ su [Invio]

...

# mkdir -p /opt/dev86 [Invio]

# make install [Invio]
```

Compilazione del kernel

La compilazione di un kernel ELKS procede in modo simile a quella di un kernel Linux. Anche in questo caso si segue lo stesso criterio usato per la compilazione degli strumenti di sviluppo, pertanto l'utente fa questo lavoro a partire dalla directory '`~/ELKS/`':

```
$ cd [Invio]

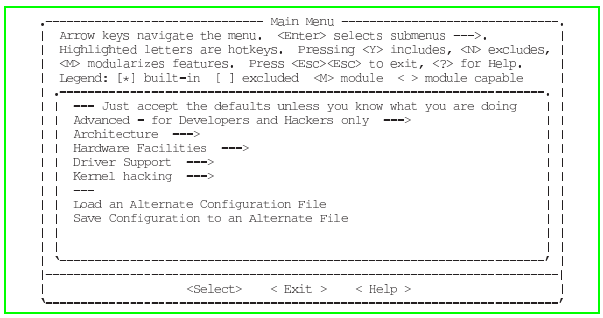
$ cd ELKS [Invio]

# tar xzvf elks-0.1.1.tar.gz [Invio]

$ cd elks-0.1.1 [Invio]
```

A questo punto si può procedere con la configurazione di ciò che si vuole nel kernel ELKS da compilare. Si può procedere con i comandi '**`make config`**' o '**`make menuconfig`**':

```
$ make menuconfig [Invio]
```



Terminata la fase di configurazione, si passa alla compilazione:

```
$ make dep [Invio]

$ make clean [Invio]

$ make Image [Invio]
```

Al termine della compilazione, se non ci sono problemi, si ottiene il file '`arch/i86/boot/Image`'.

Immagini di dischetti già pronti

Inizialmente si può cominciare tentando di avviare ELKS a partire da file-immagine già pronti. Per trasferire i file-immagine nei dischetti ci si comporta nel modo solito, esattamente come si fa per le immagini di dischetti di GNU/Linux, con la differenza che ci possono anche essere dischetti che incorporano sia il kernel, sia il file system.

Inizialmente, per fare esperimenti, è meglio usare il metodo tradizionale del kernel in un dischetto che si avvia e il file system principale in un altro, in modo da poter cambiare facilmente sia il kernel, sia il contenuto del file system.

Avvio di ELKS all'interno di DOSEMU

ELKS può essere avviato all'interno di DOSEMU. Ciò richiede un minimo di esperienza con DOSEMU, ma è utile quando si fanno delle modifiche al kernel, per fasi successive, perché l'avvio del sistema operativo è più rapido. A ogni modo, bisogna considerare che l'ambiente offerto da DOSEMU è diverso da quello reale; per esempio, se si sta modificando la mappa della tastiera, DOSEMU non va bene.

Spegnimento

Per concludere il funzionamento di un sistema ELKS può darsi che manchi sia il programma '**`halt`**', sia '**`shutdown`**'. In questa situazione, un modo possibile di agire è il seguente:

```
# sync [Invio]

# umount / [Invio]
```

Dopo questa operazione, il file system principale dovrebbe essere ancora accessibile in sola lettura.

Riferimenti

- *ELKS: the embeddable Linux kernel system*
  http://elks.sourceforge.net/
- *Source Forge: ELKS*
  http://sourceforge.net/projects/elks/
  Il sorgente più recente del kernel e degli altri componenti può essere prelevato con CVS:

```
$ cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/elks
co elks [Invio]

$ cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/elks
co elksnet [Invio]

$ cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/elks
co elkscmd [Invio]
```

- Daniele Cono D'Elia, *ELKS Embeddable Linux Kernel Subset*
  http://www.valug.it/listing/documenti/ELKS/elks.pdf

[1] **ELKS** GNU GPL

File system e dischetti 2007
File di dispositivo 2007
Sistema di avvio 2008
Bootblocks 2009
Bootkit 2011
Installazione manuale nel disco fisso 2012
Adattamento della mappa della tastiera 2013

ELKS e i programmi di servizio del sistema sono distribuiti sia in forma sorgente, sia in dischetti già pronti per l'uso e l'installazione. Tuttavia, questi dischetti non funzionano bene con tutti i tipi elaboratore, per quanto questi possano essere vecchi, o anche antichi, pertanto conviene prepararsi alla realizzazione in proprio dei dischetti, in modo da poter raggirare eventuali ostacoli imprevisti.

In generale, per tutti i lavori necessari a preparare il proprio sistema ELKS, occorre utilizzare un elaboratore funzionante con un sistema GNU/Linux.

File system e dischetti

Salvo l'uso di estensioni particolari, ELKS è in grado di accedere soltanto a file system Minix con i nomi della lunghezza massima di 14 byte.

Quando si usa un sistema GNU/Linux per inizializzare i dischetti o i file-immagine dei dischetti da usare con ELKS, bisogna tenere presente che il programma '**`mkfs.minix`**' richiede l'uso di un'opzione appropriata, altrimenti genera un file system Minix incompatibile. Nell'esempio seguente si inizializza il dischetto corrispondente al file di dispositivo '`/dev/fd0`':

```
# mkfs.minix -n 14 /dev/fd0 [Invio]
```

File di dispositivo

I file di dispositivo usati da ELKS sono simili, ma non uguali a quelli di un sistema GNU/Linux comune. La situazione più importante da osservare riguarda i file di dispositivo per l'accesso ai dischetti e ai dischi fissi. Una problema abbastanza comune riguarda l'uso di '**`rdev`**' per modificare un kernel già compilato in modo che avvii un disco diverso da quello previsto in fase di compilazione: '**`rdev`**' può essere usato anche da un sistema GNU/Linux, ma i numeri primario e secondario dei file di dispositivo non corrispondono.

Tabella u178.1. File di dispositivo principali di un sistema ELKS.

Nome	Tipo	Numero primario	Numero secondario	Sigla in esadecimale
'/dev/mem'	caratteri	1	1	0101_{16}
'/dev/kmem'	caratteri	1	2	0102_{16}
'/dev/null'	caratteri	1	3	0103_{16}
'/dev/zero'	caratteri	1	5	0105_{16}
'/dev/full'	caratteri	1	7	0107_{16}
'/dev/bda'	blocchi	3	0	0300_{16}
'/dev/bda1'	blocchi	3	1	0301_{16}
'/dev/bda2'	blocchi	3	2	0302_{16}
'/dev/bda3'	blocchi	3	3	0303_{16}
'/dev/bda4'	blocchi	3	4	0304_{16}
'/dev/bdb'	blocchi	3	64	0340_{16}
'/dev/bdb1'	blocchi	3	65	0341_{16}

Nome	Tipo	Numero primario	Numero secondario	Sigla in esadecimale
‘/dev/bdb2’	blocchi	3	66	0342_{16}
‘/dev/bdb3’	blocchi	3	67	0343_{16}
‘/dev/bdb4’	blocchi	3	68	0344_{16}
‘/dev/fd0’	blocchi	3	128	0380_{16}
‘/dev/fd1’	blocchi	3	192	$03C0_{16}$
‘/dev/tty1’	caratteri	4	0	0400_{16}
‘/dev/tty2’	caratteri	4	1	0401_{16}
‘/dev/tty3’	caratteri	4	2	0402_{16}
‘/dev/ttyS0’	caratteri	4	64	0440_{16}
‘/dev/ttyS1’	caratteri	4	65	0441_{16}
‘/dev/ttyS2’	caratteri	4	66	0442_{16}
‘/dev/lp0’	caratteri	6	0	0600_{16}
‘/dev/lp1’	caratteri	6	1	0601_{16}
‘/dev/lp2’	caratteri	6	2	0602_{16}
‘/dev/tcpdev’	caratteri	8	0	0800_{16}

I file di dispositivo ‘`/dev/bd*`’ si riferiscono all’accesso al disco fisso, attraverso il BIOS. Eventualmente, sono previsti file di dispositivo per l’accesso diretto al disco fisso, con la denominazione consueta ‘`/dev/hd*`’, anche se inizialmente il kernel non è in grado di farlo, ma ugualmente il numero primario differisce da quello usato con i sistemi GNU/Linux.

Per la ricostruzione dei file di dispositivo è disponibile uno script ‘**`MAKEDEV`**’ già pronto, con descrizioni molto chiare, ma forse conviene riepilogare i comandi per i file di dispositivo elencati nella tabella:

```
#!/bin/sh

mknod mem       c 1 1
mknod kmem      c 1 2
mknod null      c 1 3

mknod zero      c 1 5
mknod full      c 1 7

mknod bda       b 3 0
mknod bda1      b 3 1
mknod bda2      b 3 2
mknod bda3      b 3 3
mknod bda4      b 3 4

mknod bdb       b 3 64
mknod bdb1      b 3 65
mknod bdb2      b 3 66
mknod bdb3      b 3 67
mknod bdb4      b 3 68

mknod fd0       b 3 128
mknod fd1       b 3 192

mknod tty1      c 4 0
mknod tty2      c 4 1
mknod tty3      c 4 2

mknod ttyS0     c 4 64
mknod ttyS1     c 4 65
mknod ttyS2     c 4 66

mknod lp0       c 6 0
mknod lp1       c 6 1
mknod lp2       c 6 2

mknod tcpdev    c 8 0
```

Sistema di avvio

Un problema molto importante da risolvere quando si vuole mettere insieme un sistema operativo è il meccanismo di avvio. Il kernel ELKS, così come avviene nel caso di Linux, si avvia da solo, copiandolo in un dischetto come se fosse l’immagine dello stesso, offrendo così una base di partenza sicura.

Il fatto di avviare il kernel ELKS direttamente implica l’impossibilità di passargli dei parametri, pertanto bisogna utilizzare ‘**`rdev`**’ o un programma simile per definire da quale file di dispositivo deve essere innestato il file system principale:

```
# rdev file_kernel file_di_dispositivo [Invio]
```

Il programma ‘**`rdev`**’ è disponibile sia in un sistema GNU/Linux comune, sia in un sistema ELKS; quello che cambia, purtroppo, sono i numeri primario e secondario dei file di dispositivo. Per esempio, se si sta operando attraverso un sistema GNU/Linux e si vuole impostare il file ‘`elks`’, che si intende essere un kernel ELKS, occorre prima preparare il file di dispositivo appropriato, anche se questo lo si può fare in una directory temporanea:

```
# mknod /tmp/fd0 b 3 128 [Invio]

# rdev elks /tmp/fd0 [Invio]

# rdev elks [Invio]

  Root device 0x0380
```

Come si può intuire, il numero 0380_{16} rappresenta un file di dispositivo con numero primario pari a 03_{16}, ovvero 3_{10}, e numero secondario pari a 80_{16}, ovvero 128_{10}. Se invece si fa riferimento a un numero primario e secondario che in qualche modo sono previsti nel sistema, si può ottenere un’informazione un po’ confusa:

```
# mknod /tmp/bda1 b 3 1 [Invio]

# rdev elks /tmp/bda1 [Invio]

# rdev elks [Invio]

  Root device /dev/hda1
```

Evidentemente occorre fare attenzione per non confondersi.

Bootblocks

Il pacchetto Bootblocks [1] consente di avviare un sistema ELKS contenuto in un dischetto o in una partizione del disco fisso, con il kernel inserito nello stesso file system. Il pacchetto viene distribuito assieme agli strumenti di sviluppo Dev86, ma non viene compilato automaticamente assieme a quelli. Si trova precisamente nella sottodirectory ‘`bootblocks/`’ dei sorgenti di Dev86. Si compila in modo molto semplice con il comando ‘**`make`**’:

```
# cd sorgenti_dev86/bootblocks [Invio]

# make [Invio]
```

Dalla compilazione si ottengono diversi file; per quanto riguarda il problema dell’avvio di ELKS sono utili:

File	Descrizione
‘**`makeboot`**’	programma per l’installazione del settore di avvio, da usare attraverso un sistema GNU/Linux comune;
‘**`makeboot.com`**’	programma analogo a ‘**`makeboot`**’, da usare con un sistema Dos;
‘**`minix_elks.bin`**’	programma di appoggio per l’avvio di un kernel ELKS.

Per fare in modo che un dischetto, con file system Minix, contenente un sistema ELKS, completo di kernel e di programmi di servizio, occorre predisporre alcune cose come se le aspetta il sistema di avvio. Per la precisione si deve preparare la directory ‘`/boot/`’ contenente il kernel e il file ‘`minix_elks.bin`’, ma entrambi devono avere un nome appropriato; pertanto, il kernel deve essere ‘`/boot/linux`’ e il programma di appoggio (in origine ‘`minix_elks.bin`’) deve chiamarsi ‘`/boot/boot`’.

Supponendo di utilizzare un sistema GNU/Linux, supponendo di avere preparato il dischetto Minix (con i nomi al massimo di 14 byte) contenente tutto quello che serve, soprattutto con la directory

'`/boot/`' come spiegato, se questo dischetto risulta inserito nell'unità corrispondente al file di dispositivo '`/dev/fd0`', **senza essere stato innestato**, si può eseguire il comando seguente, tenendo conto che il programma '**`makeboot`**' si presume collocato in una directory prevista tra i vari percorsi della variabile di ambiente '**`PATH`**':

```
# makeboot minix /dev/fd0 [Invio]

  Wrote sector 0
  Wrote sector 1
```

Se il programma si accorge che il settore di avvio del dischetto contiene già qualcosa, si rifiuta di procedere, a meno di usare l'opzione '**`-f`**':

```
# makeboot -f minix /dev/fd0 [Invio]

  Boot block isn't empty, zap it first
  Wrote sector 0
  Wrote sector 1
```

Riquadro u178.8. Come cambiare il nome del file del kernel.

Il sistema di avvio che si ottiene con l'uso di '**`makeboot`**' e del file '`minix_elks.bin`' prevede che il kernel sia precisamente il file '`/boot/linux`', mentre forse sarebbe più appropriato '`/boot/elks`'. Per ovviare a questa piccola incoerenza, basta intervenire nel sorgente, precisamente nel file '***sorgenti_dev86***`/bootblocks/minix_elks.c`'. A un certo punto, questo file contiene le righe seguenti:

```
elks_name:
   .asciz       "linux"
   .byte        0,0,0,0,0,0,0,0
```

Per fare in modo di avviare il kernel contenuto nel file '`/boot/elks`', si deve modificare e ricompilare:

```
elks_name:
   .asciz       "elks"
   .byte        0,0,0,0,0,0,0,0
```

Per installare ELKS in una partizione del disco fisso, le cose si complicano, perché manca la possibilità di usare il programma '**`makeboot`**' con il sistema ELKS stesso e purtroppo, il programma '`makeboot.com`', funzionante con un sistema Dos, si rifiuta di intervenire in partizioni che non siano Dos. Pertanto, occorre preparare una porzione di codice da incollare poi con il programma '**`dd`**' o simile:

```
# touch avvio [Invio]

# makeboot minix avvio [Invio]

  Cannot read sector 0, clearing
  Cannot read sector 1, clearing
  Wrote sector 0
  Wrote sector 1
```

In pratica, si crea un file vuoto, in questo caso il file '`avvio`', quindi gli si inserisce il codice necessario, da incollare successivamente all'inizio della partizione da avviare.

Supponendo di avere inserito questo file in un dischetto contenente un sistema ELKS funzionante, con il quale si è riusciti a predisporre una partizione del disco fisso allo scopo di alloggiare il sistema stesso, basta copiare il file in questo modo:

```
# dd if=avvio of=/dev/bda1 [Invio]
```

Naturalmente, in questo caso si sta facendo riferimento alla prima partizione.

Bisogna ricordare che il file del kernel deve essere modificato con '**`rdev`**', in modo da utilizzare il file system contenuto nella partizione stessa.

Perché il codice iniziale della partizione venga messo in funzione, è necessario che il primo settore del disco fisso (MBR) indichi la partizione stessa come avviabile.

Nonostante tutto, può succedere che il codice inserito all'inizio della partizione venga rifiutato dal BIOS, per mancanza della firma $55AA_{16}$. In presenza di difficoltà di questo tipo, rimane la possibilità di avviare a partire da un dischetto con il kernel ritoccato attraverso '**`rdev`**' in modo da utilizzare la partizione corretta.

Bootkit

Bootkit [2] è un sistema di avvio abbastanza buono, che in particolare può passare dei parametri di avvio al kernel ELKS.

Il problema di Bootkit è che non si trovano i sorgenti.

Si può trovare Bootkit incorporato nella distribuzione EDE (*ELKS distribution edition*) e si utilizza all'interno del sistema ELKS stesso.

```
bootkit opzioni file
```

Per ottenere il risultato, occorre predisporre la directory '`/boot/`' nel file system Minix che contiene il sistema ELKS da avviare. All'interno di questa directory si colloca il file '`boot.conf`', con una serie di direttive e di solito il file '`boot.txt`', il cui contenuto deve essere mostrato in fase di avvio. A titolo di esempio, il file '`boot.conf`' potrebbe contenere le righe seguenti:

```
message=boot.txt
prompt
timeout 100
image=elks
    label=fd0
    root=0x380
image=elks
    label=fd1
    root=0x381
image=elks
    label=bda1
    root=0x301
image=elks
    label=bda2
    root=0x302
image=elks
    label=bda3
    root=0x303
image=elks
    label=bda4
    root=0x304
#other=/dev/bda1
#    label=dos
```

Inizialmente, la direttiva '**`message=boot.txt`**' dichiara di usare il file '`boot.txt`' per mostrare un messaggio all'avvio, quindi, la direttiva '**`prompt`**' fa sì che venga mostrato un invito, con il quale poter scegliere tra le diverse possibilità di avvio. La direttiva '**`timeout 100`**' fa sì che la prima delle varie modalità di avvio sia scelta in mancanza di una risposta all'avvio, entro 10 s.

Le varie direttive '**`image=elks`**' delimitano l'inizio di una sezione, distinguibile in base al valore assegnato alla direttiva '**`label`**', che dichiara l'uso del kernel contenuto nel file '`elks`', che si deve trovare nella stessa directory ('`/boot/`'). Come si vede, all'interno di queste sezioni, la direttiva '**`root`**' consente di specificare il file system che il kernel deve innestare, anche se ciò deve essere fatto indicando i numeri primario e secondario del file di dispositivo, in forma esadecimale.

Il file '`boot.txt`' potrebbe contenere il messaggio seguente:

```
Please select one of the following root file systems:

fd0     (default)
fd1
bda1
bda2
bda3
bda4
```

Si completa il lavoro mettendo nella directory '`/boot/`' il file del kernel, che in base agli esempi deve chiamarsi '`elks`', quindi occorre creare il file '`boot`', attraverso '`bootkit`':

```
# bootkit -i boot /boot/boot [Invio]
```

Si osservi che questo comando va dato all'interno di un sistema ELKS già funzionante, ma una volta creato, il file può essere copiato così in altri dischetti.

Rimane il problema della creazione di un settore di avvio, che poi cerca il programma '`/boot/boot`', il quale poi legge la configurazione del file '`/boot/boot.conf`':

```
# bootkit -i minix -b copia_settore /dev/fd0 [Invio]
```

In questo modo si installano 1024 byte di codice all'inizio del dischetto (corrispondente al file di dispositivo '`/dev/fd0`') e si salva una copia di quanto era presente prima nel file indicato come argomento dell'opzione '**`-b`**'. L'opzione '**`-b`**' è obbligatoria in questo caso, quindi, se non si vuole salvare ciò che viene sovrascritto, si può usare '`/dev/null`':

```
# bootkit -i minix -b /dev/null /dev/fd0 [Invio]
```

Teoricamente tutto questo potrebbe funzionare anche con una partizione di un disco fisso, per esempio la prima secondo il comando seguente:

```
# bootkit -i minix -b /dev/null /dev/bda1 [Invio]
```

Per funzionare, è comunque necessario che il settore iniziale del disco fisso (MBR) indichi la partizione come avviabile.

Anche Bootkit potrebbe fallire nel compito di avviare il sistema dal disco fisso, ma rimane il fatto che da dischetto è un meccanismo ottimo, che non richiede l'uso di '**`rdev`**' per dire al kernel quale disco o partizione utilizzare.

Installazione manuale nel disco fisso

L'installazione di un sistema ELKS nel disco fisso richiede di poter disporre di un dischetto funzionante e relativamente completo, possibilmente con una shell efficiente; inoltre, prima di iniziare occorre avere un quadro abbastanza chiaro di come questo dischetto è organizzato, cosa che si può fare utilizzando un sistema GNU/Linux, attraverso tutti gli strumenti a cui si è abituati.

Quando è tutto pronto, con dischetto di ELKS, o anche con un altro sistema operativo, se ciò è possibile e preferibile, occorre intervenire nella suddivisione delle partizioni del disco fisso. Considerato che l'accesso al disco avviene attraverso il BIOS, è bene che la partizione sia piccola e si trovi all'inizio o vicino all'inizio del disco stesso.

Teoricamente, dato il fatto che l'accesso avviene tramite le funzioni del BIOS, la partizione può arrivare a un massimo di 32 Mibyte, ma in pratica, può darsi che si debba ridurre ancora di più. Comunque, la partizione può anche essere relativamente grande, ma poi, quando la si va a inizializzare, bisogna creare un file system piccolo.

La partizione deve essere di tipo *old minix*, corrispondente al codice 80_{16}.

Il sistema ELKS dovrebbe disporre del programma '**`fdisk`**' per modificare l'organizzazione delle partizioni:

```
# fdisk /dev/bda [Invio]
```

Il suo utilizzo è simile a quello dello stesso programma usato nei sistemi GNU/Linux (in particolare il comando [*?*] richiama la guida degli altri comandi disponibili), ma conviene armarsi di calcolatrice per fare i conti di quanti cilindri servono per la partizioni che si vogliono creare.

Ammesso che il settore di avvio del disco fisso contenga del codice corretto, è bene ricordare di rendere avviabile la partizione che si va a creare per ospitare ELKS.

Una volta sistemate le partizioni e salvate con il comando [*w*], si può abbandonare il programma '**`fdisk`**' ([*q*]) per passare a '**`mkfs`**' con il quale creare il file system Minix:

```
# mkfs /dev/bda1 5000 [Invio]
```

In questo caso si suppone di dover inizializzare la prima partizione del primo disco fisso, con un file system di 5000 Kibyte.

Si osservi che la dimensione in settori fornita da '**`fdisk`**' rappresenta una quantità espressa in unità da 512 byte, mentre il valore che si fornisce al programma '**`mkfs`**' esprime una quantità in unità da 1024 byte. In pratica, il file system va creato con un valore che non può eccedere la metà di quanto riporta '**`fdisk`**'.

È ormai chiaro che la partizione può essere più grande del file system Minix che si va a creare. Una volta creato, è bene provare a innestarlo, per esempio con il comando seguente ammettendo che sia disponibile la directory '`/mnt/`':

```
# mount /dev/bda1 /mnt [Invio]
```

Se si ottiene una segnalazione di errore nella quale viene affermato che il file system non è valido (non è Minix), bisogna provare a inizializzarlo con una dimensione minore.

Quando si arriva al punto di essere stati capaci di innestare il file system creato nella partizione del disco fisso, si può procedere con la copia del contenuto del dischetto, ma con prudenza. Infatti, il comando '**`cp`**' potrebbe essere molto poco amichevole, inoltre, fornendogli troppi argomenti (quando si usano dei modelli la shell li espande in elenchi che possono essere anche abbastanza numerosi) può bloccarsi, assieme a tutto il sistema.

Comunque, con un po' di prudenza, si possono ricreare le directory e al loro interno vi si possono copiare i file che si trovano nel dischetto (il programma '**`cp`**' potrebbe essere capace di copiare soltanto file normali); quindi occorre riprodurre una directory '`/dev/`' con i file di dispositivo necessari e infine si può cercare di risolvere il problema dell'avvio.

Come si può comprendere molto presto, quando ci si cimenta in un lavoro di questo tipo, occorre una strategia, che può consistere nella preparazione preventiva di qualche script che faccia buona parte di questo lavoro in modo automatico.

In queste spiegazioni ci sono molte piccole cose che sono omesse, soprattutto perché le situazioni che si presentano cambiano facilmente con una grande quantità di sfumature. È evidente che si tratta di un lavoro che può affrontare solo chi ha già una buona padronanza di un sistema GNU/Linux e non si lascia scoraggiare dai piccoli fallimenti a cui si va incontro sicuramente.

Adattamento della mappa della tastiera

Un problema che può presentarsi è quello di adattare la mappa della tastiera, cosa che richiede la ricompilazione del kernel selezionando il tipo corretto per ciò che si deve usare.

Se la mappa che serve non c'è, oppure se non funziona come ci si aspetterebbe, occorre predisporre in proprio un file con un nome che corrisponda al modello '`keys-`***`xy`***`.h`', dove ***`xy`*** sono due lettere che identificano la nazionalità. Questo file, insieme agli altri delle altre mappe, va collocato nella directory '***`sorgenti_elks`***`/arch/i86/drivers/char/KeyMaps/`'. Naturalmente, il file deve avere una certa forma, che si può intuire osservando quelli già esistenti.

Bisogna tenere in considerazione il fatto che il sistema è predisposto per un elaboratore con tastiera «XT», pertanto non può esistere il tasto [*AltGr*] e nemmeno si possono attuare tutte quelle combinazioni che invece sono disponibili con un sistema GNU/Linux comune. Probabilmente si può usare con successo solo la parte alfanumerica,

i tasti freccia, [*Esc*] e probabilmente i tasti funzionali. Quando si inserisce il [*Fissamaiuscole*] non è detto che la spia corrispondente si accenda; il tasto [*BlocNum*] non funziona e probabilmente i numeri sulla tastiera numerica si ottengono inserendo il [*Fissamaiuscole*].

Dovendo predisporre o modificare una mappa che comprende anche lettere accentate e altri simboli speciali che sono al di fuori del codice ASCII tradizionale, occorre considerare che la codifica usata è quella originale degli elaboratori 8086, ovvero quella che era nota come CP 437 (si veda la sezione 47.7.7 per trovare una copia completa della codifica CP 437).

Viene proposto un esempio di mappa per la tastiera italiana, che deve corrispondere al file '***sorgenti_elks***`/arch/i86/drivers/char/KeyMaps/keys-it.h`'. Si osservi in particolare l'intestazione che serve a uno script per fare in modo che il file venga preso in considerazione durante la configurazione del kernel.

Listato u178.14. Esempio di un file per la configurazione della tastiera secondo la disposizione italiana.

```
/* Keymap:IT:Italiano:Italy     */

#ifndef __KEYMAP_IT__
#define __KEYMAP_IT__

#if defined(CONFIG_KEYMAP_IT)

/*
    \ 1 2 3 4 5 6 7 8 9 0 ' ì
       q w e r t y u i o p è +
        a s d f g h j k l ò à ù
         z x c v b n m , . -
*/

static unsigned char xtkb_scan[] = {
  0,
  033,
  '1',  '2',  '3',  '4',  '5',  '6',  '7',  '8',  '9',  '0', '\'', 0215, '\b',
 '\t',  'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  'o',  'p', 0212,  '+',
  015,
 0202,  'a',  's',  'd',  'f',  'g',  'h',  'j',  'k',  'l', 0225, 0205,
 '\\',
 0200,
 0227,
  'z',  'x',  'c',  'v',  'b',  'n',  'm',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0241, 0242, 0243, 0244, 0245, 0246, 0247, 0250, 0251, 0252,
 0205,  '?',
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    | ! " £ $ % & / ( ) = ? ^
       Q W E R T Y U I O P é *
        A S D F G H J K L ç ° §
         Z X C V B N M ; : _
*/

static unsigned char xtkb_scan_shifted[] = {
    0,
  033,
  '!',  '"', 0234,  '$',  '%',  '&',  '/',  '(',  ')',  '=',  '?',  '^', '\b',
 '\t',  'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  'O',  'P', 0202,  '*',
 '\r',
 0202,  'A',  'S',  'D',  'F',  'G',  'H',  'J',  'K',  'L', 0207, 0370,
  '|',
 0200,
  025,
  'Z',  'X',  'C',  'V',  'B',  'N',  'M',  ';',  ':',  '_',
 0201,
  '*', 0203,  ' ', 0204,
 0221, 0222, 0223, 0224, 0225, 0226, 0227, 0230, 0231, 0232,
 0204, 0213,
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    \ < > 3 4 5 6 { [ ] } ' ~
       q w e r t y u i o p [ ]
        a s d f g h j k l @ # ù
         « » c v b n m , . -
*/

static unsigned char xtkb_scan_ctrl_alt[] = {
    0,
  033,
  '<',  '>',  '3',  '4',  '5',  '6',  '{',  '[',  ']',  '}',  '''',  '~', '\b',
 '\t',  'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  'o',  'p',  '[',  ']',
 '\r',
 0202,  'a',  's',  'd',  'f',  'g',  'h',  'j',  'k',  'l',  '@',  '#',
 '\\',
 0200,
 0227,
 0256, 0257,  'c',  'v',  'b',  'n',  'm',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0241, 0242, 0243, 0244, 0245, 0246, 0247, 0250, 0251, 0252,
 0205,  '?',
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    \ 1 2 3 4 5 6 7 8 9 0 ' ì
       Q W E R T Y U I O P è +
        A S D F G H J K L ò à ù
         Z X C V B N M , . -
*/

static unsigned char xtkb_scan_caps[] = {
    0,
  033,
  '1',  '2',  '3',  '4',  '5',  '6',  '7',  '8',  '9',  '0', '\'', 0215, '\b',
 '\t',  'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  'O',  'P', 0212,  '+',
 '\r',
 0202,  'A',  'S',  'D',  'F',  'G',  'H',  'J',  'K',  'L', 0225, 0205,
 '\\',
 0200,
 0227,
  'Z',  'X',  'C',  'V',  'B',  'N',  'M',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0221, 0222, 0223, 0224, 0225, 0226, 0227, 0230, 0231, 0232,
 0204, 0213,
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

#endif

#endif
```

Come si può intuire dai nomi degli array, si distingue tra quattro situazioni: la disposizione normale, la disposizione che entra in gioco quando si preme il tasto delle maiuscole, quando si inserisce il [*Fissamaiuscole*], infine quando si preme la combinazione [*Ctrl Alt* ***x***]. In pratica, per ottenere simboli come la chiocciola e il cancelletto, occorre usare la combinazione [*Ctrl Alt ò*] e [*Ctrl Alt à*]. Per qualche motivo, non è stato possibile collocare correttamente i simboli [<] e [<], che si ottengono invece con [*Ctrl Alt 1*] e [*Ctrl Alt 2*] rispettivamente. Inoltre, sono disponibili anche le parentesi graffe, l'accento rovesciato, la tilde e le virgolette basse uncinate, come si fa di solito in un sistema GNU/Linux, ma sempre usando una combinazione del tipo [*Ctrl Alt* ***x***].

Si osservi che quanto mostrato vale come esempio e in caso di dubbio conviene verificare il funzionamento del kernel con la mappa americana standard.

Tabella u178.15. Estratto dalla codifica CP 437, per la definizione della mappa della tastiera.

Ottale	Decimale	Esadecimale	Codice corrispondente nell'insieme di caratteri universale	Aspetto
202_8	130_{10}	82_{16}	U+00E9	é
205_8	133_{10}	85_{16}	U+00E3	ã
207_8	135_{10}	87_{16}	U+00E7	ç
212_8	138_{10}	$8A_{16}$	U+00E8	è
215_8	141_{10}	$8D_{16}$	U+00EC	ì
225_8	149_{10}	95_{16}	U+00F2	ò
227_8	151_{10}	97_{16}	U+00F9	ù
234_8	156_{10}	$9C_{16}$	U+00A3	£
256_8	174_{10}	AE_{16}	U+00AB	«
257_8	175_{10}	AF_{16}	U+00BB	»

[1] **Bootblocks** GNU GPL

[2] **Bootkit** GNU GPL

Dos: introduzione 2019
Avvio del sistema 2019
Dispositivi secondo il Dos 2020
Directory, file ed eseguibili 2020
Comandi e ambiente di avvio 2022
Caratteri jolly 2023
Invito dell'interprete dei comandi 2023
Comandi interni principali 2024
Flussi standard 2028
Accesso diretto ai dispositivi 2030
Riferimenti 2030
Dos: dischi, file system, directory e file 2031
Suddivisione in partizioni 2031
Inizializzazione di un'unità di memorizzazione 2032
Etichetta di un'unità di memorizzazione 2032
Analisi e correzione del file system 2033
Copia 2033
Trasferimento del sistema 2034
Modifica delle unità 2034
Altre particolarità 2035
Dos: configurazione 2039
File «CONFIG.SYS» 2039
File «AUTOEXEC.BAT» 2042
Comandi ridondanti 2042
Localizzazione 2043
Orologio 2044
Dos: script dell'interprete dei comandi 2045
Parametri, variabili ed espansione 2045
Chiamate di altri script 2045
Strutture di controllo 2045
Comandi utili negli script 2048
Dos: gestione della memoria centrale 2051
Gestione particolare 2051
Comandi appositi 2051
Verifica 2051
FreeDOS 2053
Installazione 2053
Impostazione e configurazione 2054
RxDOS 2055
Riferimenti 2055
Progetto GNUish 2057
Programmi di servizio vari 2057
Gnuplot 2057
Spreadsheet Calculator 2058
Ispell 2058
Perl 2058
Riferimenti 2058
The valuable DOS Freeware page 2059
Introduction 2059
OS and GUI 2060
Utility 2060
Network 2061

Compilers 2062
Typesetting 2062
More Dos software sources 2062
Search engines 2063
Introduzione a ReactOS 2065
Installazione 2065
Riferimenti 2066
DOSEMU: l'emulatore di hardware DOS compatibile 2067
Predisporre un ambiente adatto al Dos all'interno di DOSEMU 2067
La configurazione di DOSEMU 2068
Installare e utilizzare il Dos 2069

Dos: introduzione

Avvio del sistema 2019
Dispositivi secondo il Dos 2020
Directory, file ed eseguibili 2020
Comandi e ambiente di avvio 2022
Caratteri jolly 2023
Invito dell'interprete dei comandi 2023
Comandi interni principali 2024
CH, CHDIR 2024
X: 2024
MD, MKDIR 2025
RD, RMDIR 2025
DIR 2025
COPY 2026
DEL, ERASE 2027
REN, RENAME 2027
SET 2027
TYPE 2028
Flussi standard 2028
SORT 2029
MORE 2029
Accesso diretto ai dispositivi 2030
Riferimenti 2030

DOS è acronimo di *Disk Operating System* e sta a indicare il nome di un sistema operativo per micro elaboratori basati su microprocessori i86, successore del vecchio CP/M. Probabilmente, data la sua estrema limitatezza, è un po' azzardato voler parlare di «sistema operativo», tanto che qualcuno lo appella: «gestore di interruzioni» (*interrupt*).

Questo sistema operativo nasce come software proprietario; tuttavia, attualmente il progetto più attivo attorno a questo tipo di sistema è FreeDOS, il cui scopo è quello di realizzarne un'edizione libera e completa.

Avvio del sistema

Un sistema Dos è composto essenzialmente da un kernel, un interprete dei comandi e da una serie di programmi di servizio. Questo concetto è analogo ai sistemi Unix, con la differenza che il kernel offre funzionalità molto scarse e solo per mezzo di interruzioni software (IRQ).

Nelle versioni proprietarie del Dos, il kernel è suddiviso in due file, che raccoglievano funzionalità distinte in base all'importanza relativa. I nomi usati sono stati differenti e nel caso di FreeDOS il kernel è contenuto tutto in un solo file (tabella u179.1).

Tabella u179.1. Comparazione tra i nomi dei file che compongono il kernel di un sistema Dos.

Microsoft	IBM	Novell, Caldera	RxDOS	FreeDOS
IO.SYS	IBMBIO.COM	IBMBIO.COM	RXDOSBIO.SYS	KERNEL.SYS
MSDOS.SYS	IBMDOS.COM	IBMDOS.COM	RXDOS.SYS	--

I file del kernel devono trovarsi nella directory radice della partizione o del dischetto per poter essere avviati. Per la precisione, l'avvio del kernel viene gestito direttamente dal codice inserito nel settore di avvio della partizione o del dischetto (512 Kibyte), che a sua volta viene avviato dal firmware (il BIOS, secondo la terminologia

specifica dell'architettura i86 e successiva).

Il kernel, dopo essere stato avviato, non attiva una procedura di avvio, ma si limita a interpretare uno script speciale, '**CONFIG.SYS**', e subito dopo avvia l'interprete dei comandi, ovvero la shell. Tradizionalmente, il programma in questione è '**COMMAND.COM**'. Secondo la tradizione, l'interprete dei comandi che viene avviato dal kernel si occupa subito di eseguire lo script '**AUTOEXEC.BAT**'. Gli script '**CONFIG.SYS**' e '**AUTOEXEC.BAT**' devono trovarsi nella directory radice del disco o della partizione da cui si avvia il sistema, ovvero quella in cui si trova già il kernel che viene avviato.

L'interprete dei comandi, '**COMMAND.COM**', è in grado di eseguire direttamente alcune funzionalità, attraverso comandi interni che non si traducono in programmi di servizio veri e propri. Tradizionalmente '**COMMAND.COM**' si colloca nella directory radice del disco o della partizione in cui si trova il kernel stesso. Ciò non è propriamente indispensabile, ma conviene attenersi a questa linea per evitare fastidi inutili.

Dispositivi secondo il Dos

I dispositivi secondo il Dos hanno un nome, composto da lettere e cifre numeriche, terminato da due punti opzionali:

nome_dispositivo [:]

Il nome in questione può essere indicato utilizzando lettere maiuscole o minuscole, senza che la cosa faccia differenza. I nomi più comuni sono elencati nella tabella u179.2. È il caso di osservare che i due punti che concludono il nome, vanno usati necessariamente quando questo viene abbinato ad altre informazioni da cui non potrebbe essere distinto (per esempio un percorso).

Tabella u179.2. Nomi dei dispositivi più comuni in Dos.

Dispositivo	Descrizione
'A:'	Disco nella prima unità a dischetti.
'B:'	Disco nella seconda unità a dischetti.
'C:'	Prima partizione Dos nel primo disco fisso.
'D:', 'E:',... 'Z:'	Partizione Dos o altro tipo di disco.
'CON:'	Console: tastiera e schermo.
'PRN:'	Porta stampante principale.
'LPT1:', 'LPT2:',...	Porte parallele.
'COM1:', 'COM2:',...	Porte seriali.

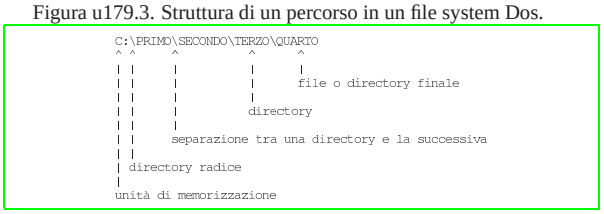
Il Dos mantiene distinti i dischi e le partizioni, nel senso che questi non devono creare una struttura unica come avviene nei sistemi Unix. Pertanto, quando si fa riferimento a un percorso di un file o di una directory, si deve tenere in considerazione anche il disco o la partizione in cui si trova.

Il modo utilizzato dal Dos per identificare i dischi e le partizioni, di fatto impedisce di accedere a questi dispositivi in modo indipendente dal file system sottostante. Per intenderci, l'«unità» 'X:' può essere una partizione Dos di un disco non meglio identificato; mentre non esiste un modo univoco per poter raggiungere il dispositivo fisico in cui si trova questo disco.

Directory, file ed eseguibili

Il Dos è nato dopo Unix e da questo sistema ha ereditato alcuni concetti elementari (forse troppo pochi). I percorsi di file e directory si separano con una barra obliqua, che però è inversa rispetto allo Unix. Anche con il Dos c'è una directory radice; tuttavia si aggiunge l'indicazione dell'unità di memorizzazione (il disco o la partizione). Si può osservare a questo proposito la figura u179.3.

Figura u179.3. Struttura di un percorso in un file system Dos.



I nomi di file e directory possono essere indicati utilizzando lettere maiuscole o minuscole, senza che la cosa possa fare differenza. Questi nomi possono essere composti utilizzando anche cifre numeriche e altri simboli (che comunque è bene usare con parsimonia).

Per la precisione, sono esclusi i simboli: '**/**', '****', '**[**', '**]**', '**<**', '**>**', '**+**', '**=**', '**;**', '**:**', '**,**', '**?**', '*****', '**{**', '**}**' e il punto che va usato esattamente come descritto nel seguito.

Tradizionalmente, il Dos utilizza un tipo di file system elementare, denominato FAT (Dos-FAT), in cui i nomi dei file e delle directory possono essere composti utilizzando al massimo 11 caratteri, di cui otto compongono un prefisso e tre un suffisso. Il prefisso e il suffisso di questi nomi appaiono uniti attraverso un punto. Per esempio: 'CIAO.COM', 'LETTERA.TXT', 'PIPPO.NB',... Questa conformazione dei nomi è una caratteristica fondamentale del Dos, da cui deriva una serie di consuetudini e di limitazioni molto importanti.

È importante osservare che non è opportuno che i nomi dei file coincidano con quelli dei dispositivi (senza i due punti finali). In pratica, non conviene creare file del tipo 'CON:', 'PRN:', ecc. Tutto dipende dal contesto, ma in generale è bene fare attenzione a questo particolare.

Come nei sistemi Unix il Dos annovera il concetto di directory corrente, a cui si aggiunge il concetto di unità di memorizzazione corrente. Infatti, la directory va collocata in un disco o in una partizione. In base a questo principio, si possono indicare dei percorsi relativi, che fanno riferimento alla posizione corrente (nell'unità di memorizzazione corrente). Tuttavia, in più, ogni unità di memorizzazione ha una sua directory corrente. Per esempio, fare riferimento a un file in una certa unità di memorizzazione '***x*:', senza specificare il percorso, significa indicare implicitamente la directory corrente di quella unità.

Per esempio, supponendo che la directory corrente dell'unità 'X:' sia 'X:\PRIMO\SECONDO\', facendo riferimento al file 'X:CIAO', si intende indicare implicitamente il file 'X:\PRIMO\SECONDO\CIAO'.

In un percorso si possono usare anche i simboli '.' e '..', con lo stesso significato che hanno in un sistema Unix: la directory stessa e la directory genitrice.

Il file system tradizionale del Dos consente di annotare solo poche informazioni per i file e le directory: la data di modifica e quattro indicatori booleani, rappresentati da altrettante lettere:

- H file o directory nascosti;
- S file o directory di sistema;
- R file o directory in sola lettura e non cancellabile;
- A file o directory da archiviare (i dati sono stati modificati).

Si tratta di attributi completamente differenti da quelli di Unix. Si può osservare in particolare la mancanza di un attributo che specifichi la possibilità di eseguire un programma o di attraversare una directory. Secondo la tradizione Dos, gli attributi vanno considerati nel modo seguente:

- A viene attivato ogni volta che il file viene scritto o modificato e serve per automatizzare i sistemi di copia periodica;

- R se attivo, il Dos non consente la scrittura o la rimozione;
- S se attivo si tratta di un file di «sistema», ma in pratica si comporta come l'attributo H;
- H se attivo si tratta di un file «nascosto», che così non dovrebbe apparire nelle liste di file e directory.

In generale, file e directory nascosti o di sistema non dovrebbero essere spostati fisicamente, nemmeno nell'ambito della stessa unità di memorizzazione. Questa esigenza nasce in particolare per i file del kernel, che non possono essere spostati se si vuole poter riavviare il sistema operativo.

Dal momento che il file system non permette di determinare se un file è un eseguibile, l'unico modo per permettere al sistema di conoscere questa caratteristica sta nell'uso di suffissi convenzionali nei nomi: i file che terminano con l'estensione '`.COM`' e '`.EXE`' sono programmi binari (la differenza tra i due tipi di estensione riguarda il formato del binario); quelli che terminano per '`.BAT`' sono script dell'interprete dei comandi ('**`COMMAND.COM`**').

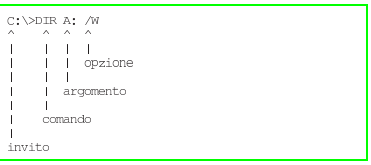
La prima stranezza che deriva da questa caratteristica del Dos sta nel fatto che per avviare un eseguibile di questi, è sufficiente indicare il nome del file senza l'estensione, che diventa così un componente opzionale agli occhi dell'utilizzatore.

Comandi e ambiente di avvio

L'interprete dei comandi tradizionale dei sistemi Dos è il programma '**`COMMAND.COM`**', che viene avviato direttamente dal kernel. '**`COMMAND.COM`**' può essere avviato più volte successive, anche se di solito ciò è di scarsa utilità, dal momento che il Dos non è un sistema operativo in multiprogrammazione. In ogni caso, quando viene avviato dal kernel, si occupa di interpretare ed eseguire lo script '**`AUTOEXEC.BAT`**' che si trova nella directory radice dell'unità di avvio.

'**`COMMAND.COM`**' mostra un invito simile idealmente a quello delle shell Unix, dopo il quale possono essere inseriti i comandi. A loro volta, questi possono essere riferiti a ***comandi interni*** corrispondenti a funzionalità offerte direttamente dall'interprete, oppure possono rappresentare la richiesta di avvio di un programma esterno.

Figura u179.4. Riga di comando.



Il Dos ha ereditato da Unix anche il concetto di variabile di ambiente. Il meccanismo è lo stesso ed è fondamentale la variabile di ambiente '**`PATH`**', con la quale si possono indicare i percorsi di ricerca degli eseguibili. Tuttavia, il Dos ha delle caratteristiche speciali, per cui, è il caso di fare alcuni esempi di comandi:

- `C:\>`**`C:\PRIMO\SECONDO.EXE`** [*Invio*]

  questo comando avvia l'esecuzione del file '`C:\PRIMO\SECONDO.EXE`';

- `C:\>`**`C:\PRIMO\SECONDO`** [*Invio*]

  questo comando potrebbe avviare l'esecuzione del primo dei file seguenti che riesce a trovare;

  - '`C:\PRIMO\SECONDO.COM`'
  - '`C:\PRIMO\SECONDO.EXE`'
  - '`C:\PRIMO\SECONDO.BAT`'

- `C:\>`**`SECONDO`** [*Invio*]

  questo comando potrebbe avviare l'esecuzione del primo dei file seguenti che dovesse riuscire a trovare, ma in mancanza può continuare la ricerca nei percorsi indicati nella variabile di ambiente '**`PATH`**'.

  - '`C:.\SECONDO.COM`'
  - '`C:.\SECONDO.EXE`'
  - '`C:.\SECONDO.BAT`'

I percorsi indicati nella variabile di ambiente '**`PATH`**' sono separati da un punto e virgola; per esempio:

```
C:\;C:\DOS;C:\FDOS\BIN
```

Di solito, il Dos dà per scontato che si cerchino gli eseguibili a cominciare dalla directory corrente. Per questo, occorre considerare che è sempre come se la variabile di ambiente '**`PATH`**' contenesse questa indicazione prima delle altre: '**`.;C:\;C:\DOS;C:\FDOS\BIN`**'. È da osservare che FreeDOS si comporta in maniera differente, in quanto richiede espressamente questa indicazione della directory corrente.

Caratteri jolly

Il Dos imita l'utilizzo dei caratteri jolly come avviene nei sistemi Unix per opera delle shell. Tuttavia, nel Dos non si tratta di un'espansione che avviene per opera della shell, ma vi deve provvedere ogni programma per conto proprio. Questo rappresenta una gravissima deficienza del Dos, che però è irrimediabile.

Su questa base, i comandi tendono a richiedere l'indicazione di un argomento che rappresenta il nome di uno o più file prima delle opzioni eventuali.

Ma c'è un altro problema. Il punto che divide in due i nomi dei file e delle directory è un muro insuperabile per i caratteri jolly.

I simboli che si possono utilizzare sono solo l'asterisco e il punto interrogativo. L'asterisco vale per una sequenza qualunque di caratteri, escluso il punto; il punto interrogativo vale per un carattere qualunque.[1]

Tabella u179.6. Alcuni esempi.

Modello	Corrispondenza
`*.*`	Corrisponde a un nome qualunque.
`*.COM`	Un nome che termina con l'estensione '`.COM`'.
`CIAO.X?X`	Tutti i nomi che iniziano per '`CIAO`' e hanno un'estensione composta da un lettera «X» iniziale e finale, senza specificare cosa ci sia al secondo posto.
`*`	Tutti i nomi che non hanno estensione (che non contengono il punto).

Invito dell'interprete dei comandi

Esiste un'altra variabile di ambiente fondamentale per il Dos. Si tratta di '**`PROMPT`**', che consente di modificare l'aspetto dell'invito dell'interprete dei comandi. La cosa funziona un po' come nelle shell Unix, per cui si assegna una stringa che può contenere dei simboli speciali, praticamente delle sequenze di escape che vengono espanse prima della visualizzazione. La tabella u179.7 riepiloga questi simboli particolari. In origine, il Dos mostrava in modo predefinito un invito simile all'esempio seguente, in cui appare solo l'unità di memorizzazione corrente:

```
C:>
```

Questo tipo di impostazione corrisponderebbe alla stringa '**`$N$G`**'. In seguito, si è passati a un invito simile al prossimo esempio, in cui si aggiunge anche l'informazione della directory corrente:

```
C:\BIN\>
```

Questo corrisponde alla stringa ‘**PG**’.

Tabella u179.7. Sequenze di escape per definire dei componenti speciali all’interno di una stringa di invito.

Simbolo	Corrispondenza
`$Q`	=
`$$`	$
`$T`	Ora corrente.
`$D`	Data corrente.
`$V`	Numero della versione.
`$N`	Lettera dell’unità corrente.
`$G`	>
`$L`	<
`$B`	|
`$H`	<*BS*> (cancella il carattere precedente)
`$E`	<*ESC*> ($1B_{16}$)
`$_`	Codice di interruzione di riga.

Cancellando il contenuto della variabile di ambiente ‘**PROMPT**’ si ripristina la stringa di invito predefinita.

Comandi interni principali

I comandi interni sono quelli che non corrispondono a programmi di servizio veri e propri, ma sono funzionalità svolte direttamente dall’interprete dei comandi. Nelle sezioni seguenti ne vengono descritti brevemente alcuni.

CH, CHDIR

```
CH [percorso]
```

```
CHDIR [percorso]
```

‘**CH**’, o ‘**CHDIR**’, è un comando interno dell’interprete dei comandi, che consente di visualizzare o di cambiare la directory corrente. È indifferente l’uso di ‘**CD**’ o di ‘**CHDIR**’; se il comando non è seguito dal percorso, si ottiene solo la visualizzazione della directory corrente. Si osservi che se si indica un percorso assoluto di unità di memorizzazione, se questa non corrisponde a quella attuale, si cambia la directory corrente di quella unità.

Segue la descrizione di alcuni esempi.

- `C:\>`**`CD`** [*Invio*]

  Visualizza la directory corrente.

- `C:\>`**`CD \TMP\LAVORO`** [*Invio*]

  Sposta la directory corrente in ‘`\TMP\LAVORO\`’.

- `C:\TMP\LAVORO>`**`CD DATI\LETTERE`** [*Invio*]

  Sposta la directory corrente in ‘`DATI\LETTERE\`’ che a sua volta discende dalla posizione iniziale precedente.

- `C:\TMP\LAVORO\DATI\LETTERE>`**`CD ..`** [*Invio*]

  Sposta la directory corrente nella posizione della directory genitrice di quella iniziale.

- `C:\TMP\LAVORO\DATI>`**`CD F:\TMP`** [*Invio*]

  Cambia la directory corrente dell’unità ‘`F:`’, senza intervenire nell’unità corrente.

X:

```
{A|B|…|Z}:
```

Il Dos gestisce le unità di memorizzazione in modo speciale. Per cambiare l’unità di memorizzazione corrente, non esiste un comando analogo a ‘**CD**’: si deve indicare il nome dell’unità a cui si vuole accedere.

Segue la descrizione di alcuni esempi.

- `C:\>`**`A:`** [*Invio*]

  Cambia l’unità di memorizzazione attuale, facendola diventare ‘`A:`’.

- `A:\>`**`F:`** [*Invio*]

  Cambia l’unità di memorizzazione attuale, facendola diventare ‘`F:`’.

MD, MKDIR

```
MD directory
```

```
MKDIR directory
```

‘**MD**’, o ‘**MKDIR**’, è un comando interno dell’interprete dei comandi, che consente di creare una directory vuota.

Segue la descrizione di alcuni esempi.

- `C:\>`**`MD LAVORO`** [*Invio*]

  Crea la directory ‘`LAVORO\`’ a partire da quella corrente.

- `C:\>`**`MD \TMP\DATA`** [*Invio*]

  Crea la directory ‘`\TMP\DATA\`’ nell’unità corrente.

- `C:\>`**`MD F:\TMP\DATA`** [*Invio*]

  Crea la directory ‘`\TMP\DATA\`’ nell’unità ‘`F:`’.

RD, RMDIR

```
RM directory
```

```
RMDIR directory
```

‘**RD**’, o ‘**RMDIR**’, è un comando interno dell’interprete dei comandi, che consente di cancellare una directory vuota.

Segue la descrizione di alcuni esempi.

- `C:\>`**`RD LAVORO`** [*Invio*]

  Cancella la directory ‘`LAVORO\`’ a partire da quella corrente.

- `C:\>`**`RD \TMP\DATA`** [*Invio*]

  Cancella la directory ‘`\TMP\DATA\`’ nell’unità corrente.

- `C:\>`**`RD F:\TMP\DATA`** [*Invio*]

  Cancella la directory ‘`\TMP\DATA\`’ nell’unità ‘`F:`’.

DIR

```
DIR [directory|file] [/P] [/W]
```

‘**DIR**’ è un comando interno dell’interprete dei comandi, che consente di visualizzare l’elenco del contenuto di una directory o l’elenco di un gruppo di file. L’argomento del comando può essere composto utilizzando caratteri jolly, secondo lo standard del Dos, ovvero i simboli ‘*****’ e ‘**?**’.

Tabella u179.8. Alcune opzioni.

Opzione	Descrizione
`/P`	Blocca lo scorrimento dell'elenco in attesa della pressione di un tasto quando questo è più lungo del numero di righe che possono apparire sullo schermo.
`/W`	Visualizza solo i nomi dei file e delle directory, senza altre informazioni, permettendo così di vedere più nomi assieme in un'unica schermata.

Segue la descrizione di alcuni esempi.

- `C:\>`**`DIR *.*`** [*Invio*]

  Visualizza l'elenco di tutti i file contenuti nella directory corrente.

- `C:\>`**`DIR ESEMPIO.*`** [*Invio*]

  Visualizza l'elenco di tutti i file il cui nome inizia per '`ESEMPIO`' e continua con un'estensione qualunque.

- `C:\>`**`DIR *.DOC`** [*Invio*]

  Visualizza l'elenco di tutti i file il cui nome termina con l'estensione '`.DOC`'.

- `C:\>`**`DIR F:\DOC\*.*`** [*Invio*]

  Visualizza l'elenco di tutti i file contenuti nella directory '`\DOC\`' dell'unità '`F:`'.

- `C:\>`**`DIR F:`** [*Invio*]

  Visualizza l'elenco di tutti i file contenuti nella directory corrente dell'unità '`F:`'.

COPY

`COPY` ***file_origine*** [***file_destinazione***] [***opzioni***]

`COPY` ***file_1*** `+` ***file_2*** [`+` …] [***file_destinazione***] [***opzioni***]

'**`COPY`**' è un comando interno dell'interprete dei comandi, che consente di copiare uno o più file (sono escluse le directory). Anche qui è consentito l'uso di caratteri jolly, ma al contrario dei sistemi Unix, i caratteri jolly possono essere usati anche nella destinazione. Il '**`COPY`**' del Dos consente anche di unire assieme più file.

Tabella u179.9. Alcune opzioni.

Opzione	Descrizione
`/V`	Fa in modo che venga verificato il risultato della copia.
`/B`	a sì che la copia avvenga in modo «binario». Questa opzione può servire quando si copia un file su un dispositivo e si vuole evitare che alcuni codici vengano interpretati in modo speciale.
`/Y`	Non chiede conferma prima di sovrascrivere i file, se questi esistono già nella destinazione.

Segue la descrizione di alcuni esempi.

- `C:\>`**`COPY ESEMPIO PROVA`** [*Invio*]

  Copia il file '`ESEMPIO`' nella directory corrente ottenendo il file '`PROVA`', sempre nella directory corrente.

- `C:\>`**`COPY C:\DOS\*.* C:\TMP`** [*Invio*]

  Copia tutto il contenuto della directory '`\DOS\`' dell'unità '`C:`' nella directory '`\TMP\`' nella stessa unità '`C:`', mantenendo gli stessi nomi.

- `C:\>`**`COPY TESTA+CORPO+CODA LETTERA`** [*Invio*]

  Copia, unendoli, i file '`TESTA`', '`CORPO`' e '`CODA`', ottenendo il file '`LETTERA`'.

- `C:\>`**`COPY *.DOC *.TXT`** [*Invio*]

  Copia tutti i file che nella directory corrente hanno un nome che termina con l'estensione '`.DOC`', generando altrettanti file, con lo stesso prefisso, ma con l'estensione '`.TXT`'.

- `C:\>`**`COPY PROVA.PRN PRN: /B`** [*Invio*]

  Copia il file '`PROVA.PRN`' nel dispositivo '`PRN:`', ovvero sulla stampante, assicurandosi che la copia avvenga senza alterare alcunché.

DEL, ERASE

`DEL` ***file***

`ERASE` ***file***

'**`DEL`**', o '**`ERASE`**', è un comando interno dell'interprete dei comandi, che consente di cancellare uno o più file (sono escluse le directory). È da considerare che i file che hanno l'attributo di sola lettura attivo, non possono essere modificati e nemmeno cancellati.

Segue la descrizione di alcuni esempi.

- `C:\TMP>`**`DEL *.*`** [*Invio*]

  Cancella tutti i file nella directory corrente.

- `C:\TMP>`**`DEL ESEMPIO.*`** [*Invio*]

  Cancella tutti i file contenuti nella directory corrente, il cui nome inizia per '`ESEMPIO`' e termina con qualunque estensione.

- `C:\TMP>`**`DEL *.BAK`** [*Invio*]

  Cancella tutti i file contenuti nella directory corrente, il cui nome termina con l'estensione '`.BAK`'.

REN, RENAME

`REN` ***file_origine*** ***nome_nuovo***

`RENAME` ***file_origine*** ***nome_nuovo***

'**`REN`**', o '**`RENAME`**', è un comando interno dell'interprete dei comandi, che consente di cambiare il nome di uno o più file (sono escluse le directory). Il primo argomento può essere un percorso relativo o assoluto, completo anche dell'indicazione dell'unità, mentre il secondo argomento è il nuovo nome, che implicitamente non può essere collocato altrove.

Segue la descrizione di alcuni esempi.

- `C:\>`**`REN ESEMPIO PROVA`** [*Invio*]

  Cambia il nome del file '`ESEMPIO`', che si trova nella directory corrente, in '`PROVA`'.

- `C:\>`**`REN *.TXT *.DOC`** [*Invio*]

  Cambia il nome di tutti i file che, nella directory corrente, hanno l'estensione '`.TXT`', trasformandoli in modo tale da avere un'estensione '`.DOC`'.

SET

`SET` [***variabile_di_ambiente=stringa***]

'**`SET`**' è un comando interno dell'interprete dei comandi che ha lo scopo di assegnare un valore a una variabile di ambiente, oppure di leggere lo stato di tutte le variabili di ambiente esistenti. Quando si assegna un valore a una variabile, questa viene creata simultaneamente; quando non si assegna nulla a una variabile, la si elimina.

Segue la descrizione di alcuni esempi.

- `C:\>`**`SET`** [*Invio*]

  Elenca le variabili di ambiente esistenti assieme al loro valore.

- `C:\>`**`SET PROMPT=$P$G$G`** [*Invio*]

  Assegna alla variabile di ambiente '**`PROMPT`**' la stringa '**`$P$G$G`**'. Questo si traduce nella modifica dell'aspetto dell'invito dell'interprete dei comandi.

- `C:\>`**`SET PATH=.;C:\BIN;D:\BIN`** [*Invio*]

  Assegna alla variabile di ambiente '**`PATH`**' la stringa '**`.;C:\BIN;D:\BIN`**'.

- `C:\>`**`SET PROMPT=`** [*Invio*]

  Elimina la variabile di ambiente '**`PROMPT`**', assegnandole la stringa nulla.

TYPE

```
TYPE file
```

'**`TYPE`**' è un comando interno dell'interprete dei comandi, che consente di leggere ed emettere il contenuto di un file attraverso lo standard output. Questo si traduce in pratica nella visualizzazione del file in questione.

Segue la descrizione di alcuni esempi.

- `C:\>`**`TYPE LETTERA`** [*Invio*]

  Emette il contenuto del file '`LETTERA`' che si trova nella directory e nell'unità corrente.

- `C:\>`**`TYPE C:\DOC\MANUALE`** [*Invio*]

  Emette il contenuto del file '`MANUALE`' che si trova nella directory '`\DOC\`' dell'unità '`C:`'.

Flussi standard

Il Dos ha ereditato da Unix anche i concetti legati ai flussi standard. In pratica, i programmi hanno a disposizione tre flussi predefiniti: uno in lettura rappresentato dallo standard input, due in scrittura rappresentati dallo standard output e dallo standard error. Il meccanismo è lo stesso di Unix, anche se non funziona altrettanto bene; infatti, non è possibile ridirigere lo standard error attraverso l'interprete dei comandi.

Secondo la tradizione delle shell Unix, la ridirezione dello standard output si ottiene con il simbolo '**`>`**' posto alla fine del comando interessato, seguito poi dal nome del file che si vuole generare in questo modo. Per esempio,

`C:\>`**`TYPE LETTERA > PRN:`** [*Invio*]

invece di visualizzare il contenuto del file '`LETTERA`', lo invia al dispositivo di stampa corrispondente al nome '`PRN:`'; inoltre,

`C:\>`**`DIR *.* > ELENCO`** [*Invio*]

invece di visualizzare l'elenco dei file che si trovano nella directory corrente, crea il file '`ELENCO`' con questi dati.

La ridirezione dello standard output fatta in questo modo, va a cancellare completamente il contenuto del file di destinazione, se questo esiste già; al contrario, si può utilizzare anche '**`>>`**', con il quale, il file di destinazione viene creato se non esiste, oppure viene solo esteso.

Lo standard input viene ridiretto utilizzando il simbolo '**`<`**', con il quale è possibile inviare un file a un comando utilizzando il flusso dello standard input.

Alcuni comandi hanno la caratteristica di utilizzare esclusivamente i flussi standard. Si parla in questi casi di programmi filtro. Il programma di servizio tipico che si comporta in questo modo è

'**`SORT`**', il quale riceve un file di testo dallo standard input e lo riordina restituendolo attraverso lo standard output. Si osservi l'esempio seguente:

`C:\>`**`SORT < ELENCO > ORDINATO`** [*Invio*]

In questo modo, '**`SORT`**' riceve dallo standard input il file '`ELENCO`' e genera attraverso la ridirezione dello standard output il file '`ORDINATO`'.

Per mettere in contatto lo standard output di un comando con lo standard input del successivo, si utilizza il simbolo '**`|`**'. L'esempio seguente mostra un modo alternativo di ottenere l'ordinamento di un file:

`C:\>`**`TYPE ELENCO | SORT > ORDINATO`** [*Invio*]

In generale, tutti i comandi che generano un risultato visuale che scorre sullo schermo, utilizzano semplicemente lo standard output, che può essere ridiretto in questo modo. Si osservi ancora l'esempio seguente che riordina il risultato del comando '**`DIR`**', mostrandolo comunque sullo schermo:

`C:\>`**`DIR *.DOC | SORT`** [*Invio*]

Nelle sezioni seguenti vengono mostrati alcuni comandi filtro.

SORT

```
SORT [opzioni] < file_da_ordinare > file_ordinato
```

Il comando '**`SORT`**', che dovrebbe corrispondere a un programma di servizio vero e proprio, riordina il file di testo che ottiene dallo standard input, generando un risultato che emette attraverso lo standard output.

Tabella u179.10. Alcune opzioni.

Opzione	Descrizione
`/R`	Riordina in modo decrescente
/+n_colonna	Riordina in base al testo che inizia a partire dalla colonna indicata come argomento (si tratta di un numero a partire da uno, per indicare la prima colonna).

L'esempio seguente emette l'elenco della directory corrente riordinato in base all'estensione, che è un'informazione collocata a partire dalla decima colonna:

`C:\>`**`DIR *.DOC | SORT /+10`** [*Invio*]

MORE

```
MORE < file_da_leggere
```

```
MORE file_da_leggere
```

Il comando '**`MORE`**' legge un file, fornito come argomento o attraverso lo standard input, mostrandolo poi sullo schermo una pagina dopo l'altra. In questo modo, è possibile leggere il contenuto dei file più lunghi delle righe a disposizione sullo schermo.

Per passare alla pagina successiva, basta premere un tasto qualunque, oppure ciò che viene indicato espressamente.

Segue la descrizione di alcuni esempi.

- `C:\>`**`DIR | MORE`** [*Invio*]

  Permette di controllare lo scorrimento a video del risultato del comando '**`DIR`**'.

- `C:\>`**`MORE LETTERA.TXT`** [*Invio*]

  Permette di controllare lo scorrimento a video del contenuto del file '`LETTERA.TXT`'.

- `C:\>`**`TYPE LETTERA.TXT | MORE`** [*Invio*]

Si ottiene lo stesso risultato dell'esempio precedente, attraverso l'uso di un condotto.

Accesso diretto ai dispositivi

Il Dos offre poche occasioni per accedere direttamente ai dispositivi. Si tratta generalmente solo della console e della porta parallela. L'esempio seguente mostra come «copiare» un file sul dispositivo di stampa, per ottenere così la sua stampa diretta:

```
C:\>COPY LETTERA PRN: [Invio]
```

La stessa cosa avrebbe potuto essere ottenuta con la ridirezione dei flussi standard:

```
C:\>TYPE LETTERA > PRN: [Invio]
```

Può essere interessante la possibilità di copiare il flusso di ingresso della console in un file:

```
C:\>COPY CON: LETTERA [Invio]
```

In questo caso, l'inserimento nel file '`LETTERA`' prosegue fino a quando viene ricevuto un codice EOF, che si ottiene qui con la combinazione di tasti [*Ctrl z*] seguita da [*Invio*].

È bene ricordare che la console, ovvero il dispositivo '`CON:`', riceve dati in ingresso attraverso la tastiera ed emette dati in uscita utilizzando lo schermo. In pratica, quando un programma attende dati dallo standard input non ridiretto, li riceve dalla console, cioè dalla tastiera; nello stesso modo, quando un programma emette dati attraverso lo standard output non ridiretto, li invia alla console, cioè sullo schermo.

Riferimenti

- *FreeDOS*
  http://www.freedos.org
- *OpenDOS Unofficial Home Page*
  http://www.deltasoft.com/opendos.htm

[1] Ci sono programmi di origine Unix, portati in Dos, che non hanno questa limitazione riferita al punto che separa l'estensione.

Dos: dischi, file system, directory e file

Suddivisione in partizioni 2031
Inizializzazione di un'unità di memorizzazione 2032
Etichetta di un'unità di memorizzazione 2032
Analisi e correzione del file system 2033
Copia 2033
DISKCOPY 2033
XCOPY 2033
Trasferimento del sistema 2034
Modifica delle unità 2034
ASSIGN 2034
JOIN 2035
SUBST 2035
Altre particolarità 2035
VERIFY 2035
APPEND 2036
ATTRIB 2036
DELTREE 2036
FIND 2036
MOVE 2037
TREE 2037
COMP e FC 2037

La gestione dei dischi, ovvero delle unità di memorizzazione di massa, è molto particolare nel Dos. In generale, si fa riferimento a queste cose attraverso un lettera che ne rappresenta il dispositivo; tuttavia, tale dispositivo può indicare un disco intero o solo una partizione, riferendosi sempre solo a dischi e partizioni Dos.

Suddivisione in partizioni

Nel Dos, tutti i dischi rimovibili, come i dischetti, non vanno suddivisi in partizioni, mentre i dischi fissi devono essere preparati in questo modo. Il programma che si usa per queste cose è '**`FDISK`**'.

Secondo il Dos, le partizioni di un disco possono essere solo quattro. Tuttavia, una partizione normale può essere suddivisa in sottopartizioni, che vengono definite tradizionalmente «estese», dove anche queste possono essere al massimo quattro. Una tale struttura ha condizionato in pratica anche altri sistemi operativi, per esempio GNU/Linux. Bisogna tenere in considerazione l'origine storica per comprendere che altri sistemi operativi possono comportarsi in modo completamente differente.

Di solito, '**`FDISK`**' ha una visione delle partizioni tutta orientata verso il Dos. Infatti, consente di creare una sola partizione primaria (ovvero una partizione normale) e altre partizioni estese (ovvero altre sottopartizioni di una seconda partizione primaria).

Bisogna considerare che il settore di avvio del Dos viene collocato nel primo settore della partizione primaria utilizzata per il Dos. In questo modo, manca la sistemazione del primo settore del disco, l'MBR, che deve contenere il codice necessario a raggiungere il settore di avvio.

```
FDISK [/MBR]
```

In generale, sembra che le varie edizioni di '**`FDISK`**' per Dos funzionino solo con il primo disco fisso.

Fondamentalmente, il programma è interattivo, per cui si avvia una maschera con la quale si interviene per mezzo di un menù. Di norma

viene consentito di cancellare le partizioni, di crearne una primaria e probabilmente una sola di estesa.

Di solito, è possibile riscrivere il settore di avvio MBR attraverso l'opzione '**/MBR**'.

Inizializzazione di un'unità di memorizzazione

L'inizializzazione di un'unità di memorizzazione, intesa come un dischetto o una partizione, si ottiene con il comando '**FORMAT**'. Questo si occupa anche di predisporre il file system Dos-FAT ed eventualmente anche di trasferire il kernel, per renderlo avviabile.

`FORMAT` ***lettera_unità*** `:` `[/N:`***settori***`]` `[/T:`***cilindri***`]` `[/S]` `[/U]`

In alcune edizioni del Dos, questo comando non inizializza l'unità di memorizzazione, ma si limita a sovrascrivere la parte iniziale. Ciò viene fatto per accelerare il procedimento e per permettere eventualmente il recupero dei dati, in caso di ripensamenti. In generale, sarebbe meglio evitare questa scorciatoia quando si tratta di unità corrispondenti ai dischetti; così, per confermare la richiesta di un'inizializzazione tradizionale, si può aggiungere l'opzione '**/U**'.

Segue la descrizione di alcuni esempi.

- `C:\>`**`FORMAT A: /U`** [*Invio*]

  Inizializza l'unità '`A:`', corrispondente a un dischetto. L'inizializzazione avviene in modo completo, essendo stata usata l'opzione '**/U**'; inoltre, dal momento che non sono state indicate altre cose, il formato usato è quello predefinito in base alla configurazione del firmware.

- `C:\>`**`FORMAT A: /N:9 /T:40 /U`** [*Invio*]

  Come nell'esempio precedente, con l'aggiunta dell'indicazione della geometria: nove settori per traccia e 40 cilindri; si sottintende la presenza di due tracce per cilindro. Pertanto, dal momento che ogni settore è di 512 byte: 2 * 40 * 9 * 512 byte = 360 Kibyte.

- `C:\>`**`FORMAT A: /N:9 /T:80 /U`** [*Invio*]

  Come nell'esempio precedente, ma con 80 cilindri: 2 * 80 * 9 * 512 byte = 720 Kibyte.

- `C:\>`**`FORMAT A: /N:15 /T:80 /U`** [*Invio*]

  Come nell'esempio precedente, ma con 15 settori per traccia: 2 * 80 * 15 * 512 byte = 1200 Kibyte.

- `C:\>`**`FORMAT A: /N:18 /T:80 /U`** [*Invio*]

  Come nell'esempio precedente, ma con 18 settori per traccia: 2 * 80 * 18 * 512 byte = 1440 Kibyte.

- `C:\>`**`FORMAT A: /S`** [*Invio*]

  Inizializza il dischetto corrispondente all'unità '`A:`', trasferendo successivamente il kernel e probabilmente anche l'interprete dei comandi ('**`COMMAND.COM`**'). Ciò avviene perché è stata usata l'opzione '**/S**'.

Etichetta di un'unità di memorizzazione

Tradizionalmente, il Dos prevede la possibilità di attribuire un nome a un'unità di memorizzazione. Questo nome viene definito solitamente «etichetta» e di fatto viene annotato come un file speciale nella directory radice (anche se poi non appare nell'elenco). Per modificare o attribuire questo nome si utilizza il comando '**LABEL**':

`LABEL` `[`***lettera_unità***`:]` `[`***nome***`]`

Se non si indica la lettera dell'unità di memorizzazione su cui intervenire, si tratta implicitamente di quella da cui è stato avviato il sistema; se non si indica il nome da attribuire, '**LABEL**' funziona in modo interattivo, chiedendo il da farsi.

In linea di principio, l'etichetta di un'unità non serve, salvo il caso di qualche programma che potrebbe utilizzarla per uno scopo particolare (per esempio i programmi di installazione per identificare i dischetti).

Esiste anche un altro comando interno per la verifica del nome di un'unità; si tratta di '**VOL**':

`VOL` `[`***lettera_unità***`:]`

Il risultato è solo l'informazione del nome stesso, con l'aggiunta del numero di serie se questo dato è disponibile.

Analisi e correzione del file system

Esistono pochi strumenti di analisi e correzione degli errori nel file system. In origine si tratta del comando '**CHKDSK**', a cui in seguito si è aggiunto '**SCANDISK**'.

`CHKDSK` ***lettera_unità*** `:` `[/F]`

'**CHKDSK**' può essere usato solo con l'indicazione di un'unità di memorizzazione; in tal caso restituisce le informazioni disponibili su questa. Se si aggiunge l'opzione '**/F**', si richiede esplicitamente la correzione, per quanto possibile, degli errori rilevati.

L'errore tipico di un file system Dos-FAT si traduce in «concatenamenti perduti», ovvero file, interi o parziali, di cui non si può conoscere il nome. Questi file potrebbero essere solo dati temporanei che è bene siano cancellati, ma questa non è la regola. '**CHKDSK**' tende a salvare questi file assegnando loro un nome più o meno casuale, lasciando all'utilizzatore l'onere di decidere cosa farne.

Copia

Nei sistemi Dos la copia è un'attività piuttosto articolata. In pratica, il comando interno '**COPY**' consente solo di copiare file puri e semplici. Per copiare un dischetto occorre il comando '**DISKCOPY**'; per copiare file e directory occorre il comando '**XCOPY**'.

DISKCOPY

`DISKCOPY` ***unità_di_origine*** `:` ***unità_di_destinazione*** `:`

'**DISKCOPY**' permette di eseguire la copia di un'unità di memorizzazione, purché si tratti di un dischetto. Il dischetto di destinazione dovrebbe essere inizializzato preventivamente.

L'unità indicata come secondo argomento, che rappresenta la destinazione, può essere la stessa di quella di origine. In questo caso, i dischetti vanno alternati nel dispositivo che li ospita, seguendo le istruzioni che dà '**DISKCOPY**' stesso.

L'esempio seguente esegue la copia di un dischetto usando lo stesso dispositivo fisico:

`C:\>`**`DISKCOPY A: A:`** [*Invio*]

XCOPY

`XCOPY` ***percorso_origine*** `[`***percorso_destinazione***`]` `[/E]` `[/S]` `[/H]` `[/V]`

'**XCOPY**' consente di copiare uno o più file assieme alla struttura di directory. In altri termini, ciò significa che è possibile copiare anche una directory intera.

Tabella u180.1. Alcune opzioni.

Opzione	Descrizione
`/S`	Copia solo le directory e le sottodirectory non vuote.
`/E`	Copia tutte le sottodirectory, anche se vuote.

Opzione	Descrizione
`/H`	Copia anche i file nascosti e di sistema.
`/V`	Verifica la copia.

L'esempio seguente copia tutta la struttura che si articola a partire dalla directory '`\PIPPO\`', nella directory '`\PAPPA\`', includendo anche i file nascosti e quelli di sistema:

`C:\>`**`XCOPY \PIPPO\*.* \PAPPA\*.* /E /S /H /V`** [*Invio*]

Trasferimento del sistema

Il Dos è un sistema operativo elementare. L'essenziale in assoluto è costituito dal kernel e dall'interprete dei comandi. Per rendere «avviabile» un dischetto o una partizione basta copiare questi file e sistemare il settore di avvio, in modo che punti correttamente al kernel. Questo si può ottenere con il comando '**`FORMAT`**', quando lo si usa con l'opzione '**`/S`**' (cosa che naturalmente implica anche l'inizializzazione dell'unità), oppure con il comando '**`SYS`**', fatto appositamente per questo:

`FORMAT` ***lettera_unità*** `: /S`

`SYS` ***lettera_unità*** `:`

A seconda del tipo di Dos, vengono copiati solo i file del kernel, oppure anche l'interprete dei comandi (necessario per avviare il sistema).

Modifica delle unità

La caratteristica del Dos per cui si distinguono le unità di memorizzazione, introduce l'esigenza di comandi particolari, che vengono descritti brevemente nelle sezioni seguenti. In particolare, si tratta della possibilità di attribuire una lettera di unità differente e di poter inserire un'unità in una directory come avviene con l'innesto di un file system nei sistemi Unix.

ASSIGN

`ASSIGN` ***lettera_unità_1*** `[:]=`***lettera_unità_2*** `[:]`

`ASSIGN /STATUS`

`ASSIGN`

Il comando '**`ASSIGN`**' permette di modifica il nome di un'unità di memorizzazione. Per ottenere questo risultato, rimane attivo come programma residente in memoria. Quando si usa senza argomenti, '**`ASSIGN`**' elimina tutte le ridefinizioni; con l'opzione '**`/STATUS`**' si ottiene lo stato attuale delle ridefinizioni; quando si indicano le lettere di unità, la prima è l'unità virtuale che viene creata come riproduzione della seconda.

Segue la descrizione di alcuni esempi.

- `C:\>`**`ASSIGN E:=A:`** [*Invio*]

  Dopo questo comando, per accedere all'unità corrispondente al primo dischetto, è possibile indicare l'unità '`E:`'.

- `C:\>`**`ASSIGN`** [*Invio*]

  Cancella tutte le ridefinizioni delle unità di memorizzazione.

JOIN

`JOIN` ***lettera_unità*** `:` ***percorso***

`JOIN` ***lettera_unità*** `: /D`

`JOIN`

Il comando '**`JOIN`**' permette di attaccare un'unità di memorizzazione in corrispondenza di un percorso (una directory). Si tratta in pratica di innestare l'unità, come avviene nei sistemi Unix.

Quando si usa '**`JOIN`**' senza argomenti, si ottiene un elenco degli innesti attivi; quando si usa l'opzione '**`/D`**', si vuole annullare il collegamento dell'unità.

Segue la descrizione di alcuni esempi.

- `C:\>`**`JOIN A: C:\MNT\A`** [*Invio*]

  Innesta l'unità '`A:`' nella directory '`C:\MNT\A\`'.

- `C:\>`**`JOIN A: /D`** [*Invio*]

  Distacca l'unità '`A:`' da un collegamento precedente.

SUBST

`SUBST` ***lettera_unità*** `:` ***percorso***

`SUBST /D`

Il comando '**`SUBST`**' permette di creare un'unità virtuale a partire da una directory di un'altra unità. In pratica, si fa in modo di permettere l'identificazione di una certa directory attraverso l'uso di una lettera di unità.

Quando si usa '**`JOIN`**' con l'opzione '**`/D`**', si vuole annullare l'unità virtuale relativa.

Segue la descrizione di alcuni esempi.

- `C:\>`**`SUBST E: C:\EXTRA\E`** [*Invio*]

  Crea l'unità virtuale '`E:`' a partire dal contenuto delle directory '`C:\EXTRA\E\`'.

- `C:\>`**`JOIN E: /D`** [*Invio*]

  Elimina l'unità virtuale '`E:`'.

Altre particolarità

La gestione del Dos di file e directory è molto strana. Nelle sezioni seguenti vengono descritti alcuni programmi tipici dei sistemi Dos riguardanti la gestione di file e directory, che non hanno trovato un'altra collocazione in questo documento, a causa della loro particolarità.

VERIFY

`VERIFY [ON|OFF]`

Il comando interno '**`VERIFY`**' permette di richiedere al sistema operativo di verificare la registrazione nelle unità di memorizzazione. Come si vede dallo schema sintattico, si attiva o si disattiva la modalità, attraverso l'uso delle parole chiave '**`ON`**' oppure '**`OFF`**'. Di solito, questa modalità è disabilitata ed è difficile definire la reale importanza di questa impostazione.

Se si usa il comando senza alcun argomento, si ottiene di sapere quale sia l'impostazione attuale.

APPEND

```
APPEND directory
```

```
APPEND ;
```

```
APPEND
```

Il comando '**APPEND**' consente di definire un percorso per la ricerca dei file di dati. In pratica, si vuole permettere ai programmi di accedere a file di dati anche quando questi si trovano fuori della collocazione prevista. '**APPEND**' può essere usato più volte, per aggiungere altre directory.

Se viene usato con l'argomento '**;**', si intende cancellare tutto l'elenco di directory di ricerca dei file di dati. Se viene usato senza argomenti, si ottiene l'elenco di queste directory.

L'esempio seguente aggiunge la directory '`C:\DATI\`' all'elenco dei percorsi di ricerca per i file di dati:

`C:\>`**`APPEND C:\DATI`** [*Invio*]

ATTRIB

```
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] file
```

Il comando '**ATTRIB**' permette di visualizzare o cambiare gli attributi del file. In pratica, utilizzando la forma '**+***x*' si attiva l'attributo ***x***, mentre con '**-***x*' si disattiva l'attributo stesso.

Segue la descrizione di alcuni esempi.

- `C:\>`**`ATTRIB *.*`** [*Invio*]

  Mostra gli attributi di tutti i file contenuti nella directory corrente.

- `C:\>`**`ATTRIB +R *.*`** [*Invio*]

  Imposta l'attributo di sola lettura per tutti i file della directory corrente.

DELTREE

```
DELTREE directory
```

Il comando '**DELTREE**' consente di eliminare una directory con tutto il suo contenuto, ricorsivamente.

L'esempio seguente elimina la directory '`C:\TEMP\CIAO\`' assieme a tutto il suo contenuto:

`C:\>`**`DELTREE C:\TEMP\CIAO`** [*Invio*]

FIND

```
FIND [opzioni] "stringa" [file]
```

Il comando '**FIND**' è uno dei più complessi nei sistemi Dos. Serve per fare una ricerca di una stringa in uno o più file, in base a quanto indicato nell'ultimo argomento, oppure all'interno dello standard input. Il risultato normale della ricerca è l'emissione delle righe che contengono la stringa cercata, assieme all'indicazione del file a cui appartengono.

Tabella u180.2. Alcune opzioni.

Opzione	Descrizione
`/V`	La ricerca avviene per le righe che **non** contengono la stringa cercata.
`/C`	Mostra solo il totale delle righe che contengono la stringa cercata.
`/N`	Mostra il numero di ogni riga che contiene la stringa cercata.

Opzione	Descrizione
`/I`	Ignora la differenza tra maiuscole e minuscole per il confronto con la stringa di ricerca. In alcune edizioni del Dos, questa modalità di funzionamento è predefinita.

Segue al descrizione di alcuni esempi.

- `C:\>`**`FIND "ciao" *.*`** [*Invio*]

  Cerca la stringa '**`ciao`**' in tutti i file della directory corrente.

- `C:\>`**`FIND "ciao" < MIO.TXT`** [*Invio*]

  Cerca la stringa '**`ciao`**' nel file '`MIO.TXT`' che viene fornito attraverso lo standard input.

MOVE

```
MOVE file_origine directory_destinazione
```

```
MOVE directory_origine directory_destinazione
```

Il comando '**MOVE**' consente di spostare file o directory in altre collocazioni. In generale, '**MOVE**' si occupa di spostare e non di rinominare i file, che invece è una funzione del comando '**REN**'.

Il comando '**MOVE**' è ambiguo e si comporta in maniera differente da una realizzazione all'altra dei sistemi Dos. In generale bisogna considerare che la destinazione può esistere o meno, implicando dei comportamenti differenti da valutare.

L'esempio seguente sposta i file e le directory contenute in '`C:\CIAO\`' nella directory '`C:\MIA\`'. Se la directory di destinazione non c'è, questa dovrebbe essere creata automaticamente, ma la cosa va verificata:

`C:\>`**`MOVE C:\CIAO\*.* C:\MIA`** [*Invio*]

TREE

```
TREE [directory]
```

Il comando '**TREE**' consente di visualizzare la struttura della directory corrente, oppure di un'altra directory indicata come argomento.

L'esempio seguente mostra la struttura della directory '`C:\CIAO\`':

`C:\>`**`TREE C:\CIAO`** [*Invio*]

COMP e FC

```
COMP file_1 file_2 [opzioni]
```

```
FC file_1 file_2 [opzioni]
```

I comandi '**COMP**' e '**FC**' permettono di verificare se due file sono identici, oppure no. Non sono molto facili da utilizzare, specialmente il primo; probabilmente vale la pena di sapere che ci sono, senza poi pretendere di sfruttare tutte le loro possibilità.

'**FC**' assomiglia molto vagamente a un comando '**diff**' di Unix, dal momento che di fronte a file di testo cerca di comprendere quale cambiamento è stato fatto. In questo senso, è probabile che '**FC**' sia il più utile tra questi due.

File «CONFIG.SYS» 2039
BREAK 2039
BUFFERS 2040
COUNTRY 2040
DEVICE, DEVICEHIGH 2040
DOS 2041
DRIVEPARM 2041
FCBS 2042
FILES 2042
INSTALL 2042
LASTDRIVE 2042
SHELL 2042
STACK 2042
File «AUTOEXEC.BAT» 2042
Comandi ridondanti 2042
Localizzazione 2043
CHCP 2043
KEYB 2043
GRAFTABL 2044
Orologio 2044

Nel Dos è un po' difficile scindere i concetti di configurazione e script, perché per configurare il sistema, occorre predisporre degli script. Si tratta dei file '`\CONFIG.SYS`' e '`\AUTOEXEC.BAT`', collocati nell'unità di avvio. Questo fatto è già stato accennato nel capitolo introduttivo; in questo si vuole approfondire un po' la cosa.

File «CONFIG.SYS»

Il file '`CONFIG.SYS`', collocato nella directory radice dell'unità di avvio, è uno script speciale avviato dal kernel prima dell'interprete dei comandi. In linea di massima, si tratta di una sequenza di direttive che occupano ognuna una riga; alcune versioni recenti del Dos consentono di suddividere le direttive in sezioni da scegliere in base a un menù iniziale.

Le direttive di '`CONFIG.SYS`' hanno la forma seguente:

```
nome=valore
```

In pratica, si assegna una stringa (senza delimitatori espliciti) a un nome che ha un significato particolare.

In questo file, vengono ignorate le righe vuote, quelle bianche e quelle che iniziano con la parola chiave '**`REM`**':

```
REM annotazione
```

È importante osservare che i nomi delle direttive non fanno differenza tra lettere maiuscole e minuscole. In generale, questo vale anche per le stringhe che vengono assegnate a questi nomi.

BREAK

```
BREAK={ON|OFF}
```

Teoricamente, questa istruzione consente di attivare o di disattivare la funzionalità abbinata alla combinazione di tasti [*Ctrl c*]. In condizioni normali, quando si assegna la parola chiave '**`ON`**', si attiva il funzionamento della combinazione [*Ctrl c*].

BUFFERS

«

```
BUFFERS=n_buffer [, n_buffer_secondari]
```

Questa istruzione consente di definire la quantità di memoria tampone per gli accessi ai dischi. Si assegnano uno o due valori numerici, separati da una virgola. Il primo valore va da 1 a 99 ed esprime il numero di aree da usare come memoria tampone; il secondo valore, facoltativo, indica delle memorie tampone secondarie, con valori che vanno da uno a otto.

COUNTRY

«

```
COUNTRY=n_codice_paese [, [n_codifica] [, file_informazioni_nazionali]]
```

Questa istruzione, attraverso quanto contenuto in un file che tradizionalmente si chiama '`COUNTRY.SYS`', permette di configurare il sistema in base alla nazionalità. Per la precisione, si può specificare un codice riferito alla nazionalità, attraverso il quale si ottiene una forma particolare per le date e gli orari, con l'aggiunta eventuale di un altro codice che specifica la codifica dei caratteri prescelta (*codepage*). La tabella u181.1 riepiloga questi codici che fanno riferimento tradizionalmente anche a paesi che non esistono più.

Si può osservare che la stringa assegnata alla direttiva '**`COUNTRY`**' può contenere l'indicazione di un file (con il percorso, completo di unità o meno). Questo file è quello che contiene poi le indicazioni relative alla nazionalità prescelta; come già accennato, di solito si tratta del file '`COUNTRY.SYS`'.

Tabella u181.1. Codici di nazionalità.

Località	Codice di nazionalità	Codifiche utili
USA	001	437, 850
Canada francese	002	863, 850
America latina	003	850, 437
Russia	007	866, 437
Olanda	031	850, 437
Belgio	032	850, 437
Francia	033	850, 437
Spagna	034	850, 437
Ungheria	036	850, 852
Jugoslavia	038	850, 852
Italia	039	850, 437
Svizzera	041	850, 437
Cecoslovacchia	042	850, 852
Regno unito	044	850, 437
Danimarca	045	850, 865
Svezia	046	850, 437
Norvegia	047	850, 865
Polonia	048	850, 852
Germania	049	850, 437
Brasile	055	850, 860
Australia	061	850, 437
Giappone	081	932, 437, 850, 942
Corea	082	934, 437, 850, 944
Cina	088	938, 437, 850, 948
Turchia	090	857, 850
Asia (inglese)	099	850, 437
Portogallo	351	850, 860
Islanda	354	850, 861
Finlandia	358	850, 437

L'esempio seguente predispone l'impostazione nazionale per l'Italia, utilizzando la codifica 850, che ha il vantaggio di essere quella più comune dei paesi che usano l'alfabeto latino:

```
COUNTRY=039,850,C:\DOS\COUNTRY.SYS
```

DEVICE, DEVICEHIGH

«

```
DEVICE=programma_di_gestione_dispositivo [opzioni]
```



```
DEVICEHIGH=programma_di_gestione_dispositivo [opzioni]
```

Si tratta di un modo per avviare un programma speciale che ha lo scopo di rimanere residente in memoria. In generale, tali programmi servono per la gestione di qualche dispositivo, indispensabile prima di avviare l'interprete dei comandi.

La differenza tra le due direttive sta nel fatto che la seconda cerca di caricare il programma nella memoria «alta».

Le opzioni riguardano il programma.

L'esempio seguente avvia il programma '**`MOUSE.SYS`**' che presumibilmente gestisce il mouse (l'opzione '**`/2`**' serve probabilmente a utilizzare il mouse collegato alla seconda porta seriale):

```
DEVICE=C:\MOUSE\MOUSE.SYS /2
```

DOS

«

```
DOS={HIGH|LOW}[,{UMB|NOUMB}]
```

```
DOS=[{HIGH|LOW},]{UMB|NOUMB}
```

Questa istruzione richiede al kernel di allocarsi nella memoria convenzionale, '**`LOW`**', o in quella alta, '**`HIGH`**'. La parola chiave '**`UMB`**' richiede di mantenere un collegamento tra la UMB e la memoria convenzionale; la parola chiave '**`NOUMB`**' fa sì che questo collegamento non abbia luogo.

DRIVEPARM

«

```
DRIVEPARM=[opzioni]
```

Si tratta di una direttiva attraverso cui si possono definire i parametri relativi ai dispositivi a blocchi, per la precisione si tratta solo di dischi, se questo può essere necessario. Le opzioni assomigliano a quelle dei programmi di servizio, iniziando con una barra obliqua normale: '**`/`*`x`*`...`**'.

Tabella u181.4. Alcune opzioni.

Opzione	Descrizione
`/d:`*`n_dispositivo_fisico`*	Consente di indicare il dispositivo attraverso un numero, da 0 a 255. Lo zero corrisponde alla prima unità a dischetti.
`/c`	Se si utilizza questa opzione, si intende che l'unità fisica è in grado di sapere se il disco è inserito o meno.
`/f:`*`n_formato`*	Stabilisce il formato del dispositivo fisico; in pratica, fissa la geometria.
`/f:0`	Dischetto 160 Kibyte, 180 Kibyte, 320 Kibyte, 360 Kibyte.
`/f:1`	Dischetto 1 200 Kibyte.
`/f:2`	Dischetto 720 Kibyte.
`/f:5`	Disco fisso.
`/f:6`	Nastro.
`/f:7`	Dischetto 1 440 Kibyte.
`/f:9`	Dischetto 2 880 Kibyte.
`/h:`*`n_testine`*	Definisce il numero di testine.
`/i`	Indica che si tratta di un dischetto da 3,5 pollici.
`/n`	Si tratta di un disco fisso.
`/s:`*`n_settori`*	Definisce il numero di settori per traccia.
`/t:`*`n_cilindri`*	Definisce il numero dei cilindri (in altri termini: il numero di tracce per faccia).

FCBS

```
FCBS=n_blocchi
```

Permette di definire il numero di blocchi di controllo dei file (*file control block*). Il valore va da 1 a 255, mentre il valore normale è di quattro blocchi.

FILES

```
FILES=n_blocchi
```

Permette di indicare il numero massimo di file aperti. Il numero che può essere assegnato va da 8 a 255. Il valore predefinito dovrebbe essere di otto file.

INSTALL

```
INSTALL=programma [opzioni]
```

Si tratta di un'istruzione con la quale si può avviare preventivamente un programma (che dovrebbe essere residente in memoria), prima dell'avvio dell'interprete dei comandi. In questo caso, a differenza della direttiva '**DEVICE**', o '**DEVICEHIGH**', si tratta di un programma normale.

Le opzioni riguardano il programma.

LASTDRIVE

```
LASTDRIVE=lettera_unità_finale
```

Consente di specificare l'ultima lettera di unità che può essere richiesta. Questo consente di risparmiare risorse, se si è consapevoli del fatto che non servono lettere oltre un certo punto. La lettera in questione può essere indifferentemente maiuscola o minuscola, senza che ciò possa fare differenza.

SHELL

```
SHELL=programma [opzioni]
```

Permette di indicare esplicitamente il programma da avviare alla fine della procedura di avvio del kernel. In generale si tratta dell'interprete dei comandi. Questa direttiva può consentire di avviare un interprete alternativo a quello normale, oppure permette di avviarlo da una collocazione insolita; inoltre permette di dare al programma in questione delle opzioni particolari.

L'esempio seguente avvia il programma '**COMMAND.COM**' che si trova nella directory '`C:\DOS\`':

```
SHELL=C:\DOS\COMMAND.COM
```

STACK

```
STACK=n_livelli [, dimensione_in_byte]
```

Con questa istruzione è possibile fissare la dimensione dello stack, utilizzando valori da 8 a 64, oltre allo zero. Il valore dopo la virgola indica la dimensione in byte di ogni livello dello stack. In questo caso i valori vanno da 32 a 512.

File «AUTOEXEC.BAT»

Il file '`AUTOEXEC.BAT`' collocato nella directory radice dell'unità di avvio, è inteso essere uno script che viene eseguito dall'interprete dei comandi, '**COMMAND.COM**', dopo l'avvio del sistema.

Questo script viene realizzato normalmente in modo sequenziale, senza strutture di controllo. In generale è importante per due cose: impostare alcune variabili di ambiente fondamentali, per esempio '**PATH**'; avviare dei programmi che poi restano residenti in memoria, quando questo non si ottiene già attraverso il file '`\CONFIG.SYS`'.

Comandi ridondanti

Anche nel Dos è molto importante l'uso delle variabili di ambiente. È già stato mostrato il comando '**SET**', attraverso il quale si impostano o si annullano le variabili di ambiente:

```
SET nome_variabile=stringa_assegnata
```

Alcune variabili hanno un'importanza particolare, per cui esiste un comando interno apposito (dell'interprete dei comandi), che serve a inizializzarle senza nemmeno l'uso del comando '**SET**'.

- ```
  PROMPT stringa_di_invito
  ```

  Il comando interno '**PROMPT**' rappresenta un modo alternativo per impostare la variabile di ambiente con lo stesso nome. Se si usa il comando senza l'argomento, si ripristina l'invito predefinito.

- ```
  PATH [percorsi_degli_eseguibili]
  ```

  Il comando interno '**PATH**' rappresenta un modo alternativo per impostare la variabile di ambiente con lo stesso nome. Se non si indica l'argomento, si ottiene la visualizzazione dell'elenco dei percorsi attivo.

Esistono altri comandi particolari che si sovrappongono alle istruzioni del file '`CONFIG.SYS`'.

- ```
  BREAK [ON|OFF]
  ```

  Abilita o disabilita la funzionalità abbinata alla combinazione di tasti [*Ctrl c*]. Utilizzando il comando senza argomento, si ottiene la visualizzazione dello stato attuale.

Localizzazione

La localizzazione del Dos si riduce alla configurazione della mappa della tastiera e alla definizione dell'insieme di caratteri. L'insieme di caratteri dipende dalla scelta della nazionalità, fatta nel file '`CONFIG.SYS`', attraverso la direttiva '**COUNTRY**'.

Nelle sezioni seguenti vengono mostrati alcuni comandi utili per le impostazioni che riguardano la localizzazione.

CHCP

```
CHCP [n_codifica]
```

Si tratta di un comando interno dell'interprete dei comandi che interviene nella definizione della codifica utilizzata. In pratica, se si utilizza senza argomenti, mostra il numero della codifica attiva; se si indica un numero come argomento, cambia la codifica attiva, purché questa sia una di quelle ammissibili in base alla nazionalità stabilita con la direttiva '**COUNTRY**' nel file di configurazione '`CONFIG.SYS`'.

L'esempio seguente fa in modo che sia attivata la codifica corrispondente al numero 850:

```
C:\>CHCP 850 [ Invio ]
```

KEYB

```
KEYB [sigla_nazionale[, [n_codifica][, file_informazioni_tastiere]]]
```

'**KEYB**' è un comando esterno che consente di cambiare la configurazione della tastiera secondo alcuni modelli di nazionalità predefiniti. La sigla nazionale è un codice di due lettere che, assieme alla nazionalità, dovrebbe indicare anche la lingua utilizzata. La tabella u181.6 elenca queste sigle.
```

Tabella u181.6. Sigle nazionali-linguistiche per l'impostazione della mappa della tastiera.

Sigla	Corrispondenza
US	USA (predefinito)
FR	Francia
GR	Germania
IT	Italia
SP	Spagna
UK	Gran Bretagna
PO	Portogallo
SG	Svizzera tedesca
SF	Svizzera francese
DK	Danimarca
BE	Belgio
NL	Olanda (Nederland)
NO	Norvegia
LA	America latina
SV	Svezia
SU	Finlandia (Suomi)
CF	Canada francese

Segue la descrizione di alcuni esempi.

- `C:\>`**`KEYB`** [*Invio*]

  Mostra la configurazione attuale.

- `C:\>`**`KEYB IT`** [*Invio*]

  Predispone la mappa dei tasti per la disposizione italiana.

- `C:\>`**`KEYB IT,850,C:\DOS\KEYBOARD.SYS`** [*Invio*]

  Predispone la mappa dei tasti per la disposizione italiana, specificando l'uso della codifica 850 e del file '`C:\DOS\KEYBOARD.SYS`' per trovare le impostazioni standard delle tastiere.

GRAFTABL

```
GRAFTABL [n_codifica]
```

```
GRAFTABL /STATUS
```

'**`GRAFTABL`**' è un comando esterno che consente di cambiare la codifica per i caratteri visualizzati sullo schermo. L'opzione '**`/STATUS`**' permette di conoscere la situazione attuale, mentre l'indicazione di un numero di codifica cambia l'impostazione.

L'esempio seguente imposta l'uso della codifica 850:

`C:\>`**`GRAFTABL 850`** [*Invio*]

Orologio

Il Dos consente di accedere all'orologio dell'elaboratore, per leggere la data e l'ora, o per cambiare tali informazioni. In generale, il Dos non prevede la gestione di un orologio hardware allineato al tempo universale; pertanto, l'orologio hardware deve corrispondere necessariamente all'ora locale, lasciando all'utente il problema legato alle variazioni dell'ora estiva.

I comandi per accedere all'orologio sono '**`DATE`**' e '**`TIME`**':

```
DATE [data]
```

```
TIME [orario]
```

Se non si indica la data o l'orario, viene mostrato quello attuale e viene richiesto all'utente di modificarlo o di confermarlo.

Il modo in cui va scritta da data o l'ora, dipende dalla localizzazione. Per conoscere quello giusto, basta osservare in che modo vengono visualizzate tali informazioni.

Dos: script dell'interprete dei comandi

Parametri, variabili ed espansione 2045
Chiamate di altri script 2045
Strutture di controllo 2045
IF 2046
FOR 2046
GOTO 2047
Emulazione di un ciclo iterativo 2047
Comandi utili negli script 2048
REM 2048
ECHO 2048
PAUSE 2048
CLS 2049
CHOICE 2049

Uno script dell'interprete dei comandi, conosciuto solitamente con il nome di file *batch*, potrebbe essere definito come un file di testo normale in cui può essere indicato un elenco di comandi da eseguire. Tuttavia, questi script consentono l'uso anche di strutture di controllo elementari, per cui si possono realizzare dei programmi molto semplici, senza troppe pretese.

È interessante osservare che questi script vengono individuati solo attraverso l'estensione che ha il nome: '`.BAT`'. Inoltre, non esiste la necessità di renderli «eseguibili» come si fa nei sistemi Unix.

Parametri, variabili ed espansione

Gli script dell'interprete dei comandi hanno accesso agli argomenti che vengono loro forniti. Si possono gestire solo nove di questi argomenti alla volta, attraverso i parametri posizionali relativi, da '**`%1`**' a '**`%9`**'. Come avviene nelle shell Unix, è disponibile il comando interno '**`SHIFT`**' per fare scorrere in avanti gli argomenti nei parametri disponibili.

Bisogna ricordare che in Dos i caratteri jolly non vengono espansi dalla shell, per cui la limitazione a soli nove parametri posizionali, non dovrebbe costituire un problema.

Nell'ambito di uno script possono essere dichiarate e utilizzate delle variabili di ambiente. È già stato mostrato in precedenza l'uso del comando '**`SET`**' per impostare o eliminare le variabili di ambiente. Per fare riferimento al contenuto di una variabile, si usa la notazione seguente:

```
%nome_variabile%
```

L'esempio seguente rappresenta il caso tipico di estensione di un percorso di ricerca degli eseguibili, quando si ritiene che la variabile '**`PATH`**' sia già stata usata:

```
SET PATH=%PATH%;C:\PIPPO
```

Chiamate di altri script

Tradizionalmente, il Dos ha un baco molto grave, ormai divenuto una caratteristica fondamentale, riguardante l'avvio di script all'interno di altri script. In generale, quando si chiama un programma che in realtà corrisponde a uno script, al termine di questo non riprende l'esecuzione di quello chiamante. Per ottenere la ripresa dell'interpretazione dello script di partenza occorre usare il comando speciale '**`CALL`**'.

```
CALL nome_script [argomenti_dello_script]
```

Strutture di controllo

Le strutture di controllo per la programmazione attraverso gli script dell'interprete dei comandi sono molto limitate. È disponibile una struttura condizionale semplificata e un ciclo di scansione di file, che vengono descritti brevemente.

IF

```
IF [NOT] ERRORLEVEL valore_di_uscita_ultimo_comando comando
```

```
IF [NOT] stringa_1==stringa_2 comando
```

```
IF [NOT] EXIST file comando
```

La struttura condizionale degli script dell'interprete dei comandi Dos è in pratica un comando interno dello stesso interprete. Come si può vedere dagli schemi sintattici, viene fornita una condizione che può essere invertita con la parola chiave '**NOT**' e il risultato è solo l'esecuzione di un altro comando se la condizione risulta vera.

Nel primo caso, la condizione si riferisce alla verifica del valore di uscita dell'ultimo comando eseguito. La condizione si verifica se il numero indicato è inferiore o uguale al valore restituito effettivamente da tale comando; nel secondo, la condizione si verifica se le due stringhe (non delimitate) sono identiche; nel terzo si verifica la condizione se il file indicato esiste effettivamente.

Segue la descrizione di alcuni esempi.

- `IF ERRORLEVEL 1 GOTO :errore`

  Se il comando precedente ha restituito un valore maggiore o uguale a uno, salta all'etichetta '**:errore**'.

- `IF %1==ciao ECHO L'argomento è corretto`

  In questo caso, se l'espansione del parametro '**%1**', corrispondente al primo argomento ricevuto all'avvio, si traduce nella stringa '**ciao**', viene emesso un messaggio per mezzo del comando '**ECHO**'.

- `IF %1x==x ECHO L'argomento è mancante`

  Quello che si vede è il trucco necessario per poter verificare se un parametro contiene la stringa nulla: si aggiunge una lettera, in questo caso una «x», verificando che la corrispondenza avvenga solo con la stessa lettera.

- `IF NOT EXIST LETTERA.TXT ECHO Scrivi! > LETTERA.TXT`

  Qui, se non esiste il file 'LETTERA.TXT' nella directory corrente, questo file viene creato attraverso il comando '**ECHO**' che invia il suo standard output verso un file con lo stesso nome.

FOR

```
FOR [%]%x IN (nome…) DO comando [argomenti_del_comando]
```

Si tratta di un comando interno che svolge un ciclo di scansione di un gruppo di nomi, generalmente file, attraverso il quale viene creato un parametro variabile speciale, il cui nome si compone di una sola lettera, a cui viene assegnato a ogni ciclo uno dei nomi contenuti tra parentesi tonde. A ogni ciclo viene eseguito il comando, che a sua volta può fare uso del parametro.[1]

Quando viene usato all'interno di uno script dell'interprete dei comandi, il parametro viene indicato con due simboli di percentuale ('**%%**x'); al contrario, se il comando viene impartito dalla riga di comando, se ne usa uno solo.

Segue la descrizione di alcuni esempi.

- `FOR %A IN (uno due tre) DO ECHO %A`

  In questo modo, si ottiene la visualizzazione delle parole '**uno**', '**due**' e '**tre**'. In pratica, è come se fosse stato fatto:

- ```
  ECHO uno
  ECHO due
  ECHO tre
  ```

  Volendo fare la stessa cosa dalla riga di comando, è necessario il raddoppio del simbolo '**%**':

  `C:\>FOR %%A IN (uno due tre) DO ECHO %%A` [*Invio*]

- `FOR %A IN (*.TMP *.BAD) DO DEL %A`

  Cancella, uno a uno, tutti i file che terminano con le estensioni '.TMP' e '.BAD'.

GOTO

```
GOTO etichetta
```

Gli script dell'interprete dei comandi dispongono dell'istruzione di salto incondizionato, non avendo di meglio. Anche questa istruzione può essere presa come un comando interno dell'interprete, con la differenza che non c'è modo di utilizzarlo al di fuori di uno script.

Nel corso di uno script del genere, possono apparire delle righe che contengono solo un'etichetta, nella forma:

```
:nome_etichetta
```

La posizione corrispondente a queste etichette può essere raggiunta con il comando '**GOTO**', che può fare riferimento solo al nome dell'etichetta, oppure a tutta l'etichetta, includendo anche i due punti.

Segue la descrizione di alcuni esempi.

- ```
  IF EXIST LETTERA.TXT GOTO riprendi
  ECHO Il file LETTERA.TXT è assente
  :riprendi
  ```

  In questo esempio, se il file 'LETTERA.TXT' esiste, si salta all'etichetta '**:riprendi**'; altrimenti si esegue il comando '**ECHO**'.

- ```
  IF EXIST LETTERA.TXT GOTO :riprendi
  ECHO Il file LETTERA.TXT è assente
  :riprendi
  ```

  Esattamente come nell'esempio precedente, con la differenza che il comando '**GOTO**' indica l'etichetta con i suoi due punti iniziali.

Emulazione di un ciclo iterativo

Dal momento che non è disponibile una struttura di controllo per il ciclo iterativo, questo può essere ottenuto solo attraverso l'uso del comando '**GOTO**'. Vale la pena di mostrare in che modo si può ottenere tale risultato.

```
:etichetta_di_ingresso
IF condizione GOTO :etichetta_di_uscita
    …
    …
    …
    GOTO etichetta_di_ingresso
:etichetta_di_uscita
```

```
:etichetta_di_ingresso
    …
    …
    …
IF condizione GOTO :etichetta_di_ingresso
:etichetta_di_uscita
```

I due modelli sintattici mostrano due esempi di cicli iterativi. Nel primo caso si verifica una condizione, in base alla quale si decide se proseguire o se terminare il ciclo; nel secondo si esegue una volta il
```

ciclo e quindi si verifica una condizione per decidere se ripeterlo o se uscire.

Comandi utili negli script

Alcuni comandi sono particolarmente utili all'interno di script dell'interprete dei comandi. Vengono descritti brevemente nelle sezioni seguenti.

REM

```
REM commento
```

I commenti negli script dell'interprete dei comandi si indicano attraverso un comando apposito: '**REM**'. Il funzionamento è evidente: tutto quello che segue il comando, fino alla fine della riga, viene ignorato.

Alcune edizioni del Dos hanno introdotto anche l'uso del punto e virgola, come simbolo per indicare l'inizio di un commento. Segue un esempio tipico di utilizzo di questo comando:

```
REM
REM (c) 2000 Pinco pallino
REM
```

ECHO

```
ECHO [ON|OFF]
```

```
ECHO stringa
```

Il comando interno '**ECHO**' ha un significato duplice: da una parte consente di visualizzare un testo; dall'altra controlla la visualizzazione dei comandi contenuti in uno script. Infatti, si distingue il fatto che l'eco dei comandi sia attivo o meno. utilizzando il comando '**ECHO**' senza argomenti, si ottiene l'informazione sul suo stato di attivazione. Di solito si disattiva l'eco dei comandi negli script.

Per disattivare l'eco di un comando particolare, senza disattivare l'eco in generale, basta inserire inizialmente il simbolo '@'.

Segue la descrizione di alcuni esempi.

- `@ECHO OFF`
  Disattiva l'eco dei comandi, facendo in modo che anche questo comando non venga visualizzato (si usa per questo il simbolo '@').
- `ECHO Premi un tasto per continuare`
  Mostra un messaggio per spiegare come comportarsi.

PAUSE

```
PAUSE
```

Il comando interno '**PAUSE**' sospende l'esecuzione di uno script in attesa della pressione di un tasto. Il comando emette attraverso lo standard output un messaggio di avvertimento in tal senso. Di solito, per evitare di vedere tale messaggio, si ridirige lo standard output in un file nullo.

L'esempio seguente, prima mostra un messaggio in cui si avverte che per proseguire occorre premere un tasto, quindi si usa il comando '**PAUSE**' che sospende l'esecuzione dello script, senza però mostrare altri messaggi:

```
ECHO Premere un tasto per proseguire
PAUSE > C:\NULL
```


CLS

```
CLS
```

Il comando interno '**CLS**' ripulisce lo schermo. Si utilizza senza argomenti.

CHOICE

```
CHOICE [opzioni] [testo_di_invito]
```

Il comando '**CHOICE**' serve a presentare una richiesta per l'inserimento di una lettera, tra un elenco determinato. La pressione del tasto corrispondente alla lettera scelta, da parte dell'utilizzatore, provoca la conclusione del funzionamento di '**CHOICE**' che restituisce un valore corrispondente alla scelta: zero per la prima lettera, uno per la seconda,...

Si osservi che l'ultimo argomento rappresenta un messaggio che serve all'utente per comprendere il senso della scelta che sta facendo.

Tabella u182.15. Alcune opzioni.

Opzione	Descrizione
`/C:`*lettera*[*lettera*...]	Permette di fissare l'elenco di lettere che possono essere usate nella risposta. Se non si indica questa opzione, la scelta avviene solo tra '**y**' e '**n**'.
`/N`	Questa opzione fa in modo di escludere la visualizzazione delle lettere che possono essere scelte. In questo modo si fa affidamento esclusivamente sul testo indicato come ultimo argomento.
`/S`	Distingue tra maiuscole e minuscole per quanto riguarda le lettere tra cui scegliere.

L'esempio seguente, salta a un punto differente dello script in base alla scelta di una lettera da «a» a «f». Si osservi che non sarebbe possibile eseguire l'analisi secondo una sequenza differente, perché '**IF ERRORLEVEL**' prende in considerazione tutti i valori di uscita maggiori o uguali a quanto indicato nella condizione.

```
CHOICE /C:abcdef Inserisci una lettera
IF ERRORLEVEL 5 GOTO :f
IF ERRORLEVEL 4 GOTO :e
IF ERRORLEVEL 3 GOTO :d
IF ERRORLEVEL 2 GOTO :c
IF ERRORLEVEL 1 GOTO :b
IF ERRORLEVEL 0 GOTO :a
…
```

[1] Questo parametro assomiglia a una variabile di ambiente, ma non si comporta allo stesso modo. Si tratta di una particolarità del comando '**FOR**'.

Dos: gestione della memoria centrale

Gestione particolare 2051
Comandi appositi 2051
Verifica 2051

Quando è nato il Dos non si prevedeva l'uso di memoria centrale oltre il singolo mebibyte (1 Mibyte). In base a questa considerazione veniva articolata l'architettura hardware degli elaboratori «XT» e poi «AT», dove si prevedeva l'uso di un massimo di 640 Kibyte di memoria centrale, riservando la parte successiva, fino alla fine di 1 Mibyte, per la memoria video e altri dispositivi fisici.

In questo senso, il Dos tradizionale può operare con un massimo di 640 Kibyte di memoria centrale; per sfruttarne di più occorrono degli accorgimenti non facili da applicare.

Gestione particolare

Per sfruttare la memoria oltre il primo mebibyte, si fa uso normalmente di due programmi, avviati attraverso '`CONFIG.SYS`', prima ancora dell'interprete di comandi. Si tratta di '**`HIMEM.SYS`**' e di '**`EMM386.EXE`**'. In generale, le cose si fanno nel modo seguente:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE
```

Il primo dei due programmi può essere utilizzato a partire da architetture i286, mentre il secondo si può inserire solo a partire da architetture i386.

'**`HIMEM.SYS`**' è in grado di utilizzare solo una piccola parte di memoria aggiuntiva, mentre '**`EMM386.EXE`**' permette teoricamente di sfruttare tutto il resto.

In generale, è molto difficile la gestione ottimale della memoria centrale, perché le applicazioni si comportano in maniera differente. Di solito si possono solo fare dei tentativi.

Comandi appositi

Per sfruttare la memoria centrale che supera la soglia convenzionale, sono disponibili alcuni comandi specifici. In generale, si comincia dalla configurazione con il file '`CONFIG.SYS`': dopo l'attivazione dei gestori speciali della memoria, è possibile indicare di collocare parte dell'interprete dei comandi e dello spazio richiesto dai programmi residenti in memoria, oltre il limite della memoria convenzionale:

```
DOS=HIGH,UMB
```

In seguito, sempre nell'ambito del file '`CONFIG.SYS`', si può richiedere esplicitamente l'avvio di programmi nella memoria alta attraverso la direttiva '**`DEVICEHIGH`**', come si vede nell'esempio seguente:

```
DEVICEHIGH=C:\MOUSE\MOUSE.SYS /2
```

Per quanto riguarda i programmi avviati attraverso l'interprete dei comandi, è disponibile il comando '**`LH`**', ovvero '**`LOADHIGH`**':

`LH` ***programma*** [***argomenti_del_programma***]

`LOADHIGH` ***programma*** [***argomenti_del_programma***]

Per esempio, si potrebbe tentare di avviare in questo modo il programma di gestione della tastiera:

`C:\>`**`LH KEYB IT`** [*Invio*]

Verifica

Il Dos offre un solo programma molto semplice per la verifica dell'utilizzo della memoria: '**`MEM`**'.

MEM [*opzioni*]

Se '**MEM**' viene usato senza opzioni, visualizza brevemente la quantità di memoria utilizzata rispetto al totale disponibile. È interessante l'opzione '**/CLASSIFY**', attraverso la quale è possibile distinguere l'utilizzo della memoria da parte dei programmi residenti; inoltre è interessante l'opzione '**/FREE**', con cui si hanno informazioni dettagliate sulla memoria libera.

Le opzioni disponibili del comando '**MEM**' variano molto da una realizzazione all'altra. In generale conviene verificare prima di utilizzarlo, per conoscere le possibilità effettive.

FreeDOS

Installazione 2053
Impostazione e configurazione 2054
RxDOS 2055
Riferimenti 2055

FreeDOS [1] è il nome di un progetto per la realizzazione di un sistema operativo libero compatibile con il Dos. Il Dos, per quanto limitato, ha delle caratteristiche che lo possono rendere ancora interessante per elaboratori con architettura i86 particolarmente poveri di risorse, come nel caso dei sistemi cosiddetti *embedded*.

Installazione

L'installazione della distribuzione standard di FreeDOS è abbastanza semplice. Si parte da un dischetto di avvio, con il quale si predispone la partizione e la si inizializza, quindi si prosegue con il programma di installazione che chiede l'inserimento dei dischetti successivi. La riproduzione del dischetto di avvio a partire dalla sua immagine avviene come al solito attraverso il programma '**RAWRITE.EXE**', oppure per mezzo di un sistema Unix nei modi già mostrati per GNU/Linux e altri sistemi simili.

```
C:\>RAWRITE FULL.BIN A:
```
[*Invio*]

L'esempio mostra l'uso di '**RAWRITE.EXE**' per ottenere un dischetto dall'immagine rappresentata dal file '`FULL.BIN`'.

La distribuzione standard di FreeDOS si compone di un file-immagine del dischetto di avvio, che potrebbe chiamarsi '`FULL.BIN`', e da una serie di file con estensione '`.ZIP`' che servono per ottenere i dischetti successivi. Ognuno di questi file compressi rappresenta il contenuto di un dischetto, che quindi deve essere prima estratto:

```
C:\>A:
```
[*Invio*]

```
A:\>UNZIP C:\TMP\BASE1.ZIP
```
[*Invio*]

L'esempio mostra in breve il procedimento: ci si sposta nell'unità '`A:`' e da lì si estrae il file compresso che probabilmente si trova da qualche parte nel disco fisso.

Questi file compressi rappresentano una raccolta di applicativi e hanno una struttura particolare che viene descritta nel seguito.

- ***nome_raccolta*.1**

  L'archivio compresso deve contenere un file che rappresenta il nome della raccolta, con un'estensione numerica. La raccolta potrebbe essere suddivisa in più archivi ed è per questo che si usa l'estensione numerica, che indica il numero di sequenza dell'archivio nell'ambito della raccolta.

  Il file contiene l'elenco dei pacchetti contenuti, con l'indicazione dell'opzione di installazione predefinita o meno. Si osservi l'estratto seguente (la lettera «Y» rappresenta la conferma all'installazione predefinita):

  ```
  asgn14x: Y
  attr063x: Y
  bwb210x: Y
  choic20x: Y
  ```

- ***nome_raccolta*.END**

  Si tratta di un file vuoto, che rappresenta la conclusione della raccolta, nel senso che non ci sono altri dischetti ulteriori.

- ***nome_pacchetto*.LSM**

  Si tratta di un file che descrive un pacchetto applicativo. Quello che segue è l'esempio del contenuto del file '`DELTR10X.LSM`':

```
Begin3
Title:          deltree
Version:        1.02b
Entered-date:   27 Jul 1999
Description:    Delete a directory and all directories under it
Keywords:       freedos delete
Author:         raster@highfiber.com
Maintained-by:  raster@highfiber.com
Primary-site:   http://www.highfiber.com/~raster/freeware.htm
Alternate-site: www.freedos.org
Original-site:  http://www.highfiber.com/~raster/freeware.htm
Platforms:      dos
Copying-policy: GPL
End
```

- ***nome_pacchetto*`.ZIP`**

  Si tratta dell'archivio compresso che contiene i file dell'applicativo. In base alla struttura standard di FreeDOS, potrebbe distribuirsi nelle directory '`BIN\`', '`DOC\`' e '`HELP\`'.

Dopo aver preparato i dischetti, si può procedere con l'avvio del sistema attraverso il dischetto di avvio; quindi si passa a predisporre la partizione:

`A:\>`**`FDISK`** [*Invio*]

Purtroppo, il kernel di FreeDOS non è in grado di gestire partizioni più grandi di 512 Mibyte, per cui occorre tenerne conto durante l'uso di '**`FDISK`**'. Dopo aver preparato la partizione la si inizializza:

`A:\>`**`FORMAT C: /U`** [*Invio*]

Successivamente si trasferisce il sistema, con il comando '**`SYS`**':

`A:\>`**`SYS C:`** [*Invio*]

Infine si avvia il programma di installazione che provvede a chiedere la sostituzione dei dischetti:

`A:\>`**`INSTALL`** [*Invio*]

Impostazione e configurazione

Da quanto è stato descritto sull'installazione di FreeDOS si intende che, pur trattandosi di un sistema Dos, si cerca di introdurre qualche buona idea proveniente da Unix. In particolare, è prevista una struttura per la collocazione dei file:

- '`BIN\`' per contenere i file eseguibili;
- '`DOC\`' per contenere la documentazione che si articola in altre sottodirectory successive, come avviene con GNU/Linux
- '`HELP\`' per contenere i file della guida interna relativa.

Questa struttura potrebbe essere collocata anche a partire da un punto differente della radice dell'unità, in base alle scelte fatte in fase di installazione. In ogni caso, occorre poi predisporre coerentemente alcune variabili di ambiente: '**`PAGER`**' per indicare il programma da utilizzare per lo scorrimento dei file delle guide; '**`HELPPATH`**' per indicare la directory contenente i file delle guide; '**`EMACS`**' per indicare la directory contenente i file di Emacs.

In condizioni normali, gli applicativi FreeDOS vengono installati a partire dalla directory '`\FDOS\`', per cui la configurazione si traduce nelle istruzioni seguenti nel file '`AUTOEXEC.BAT`':

```
SET PAGER=MORE
SET HELPPATH=C:\FDOS\HELP
SET EMACS=C:\FDOS\EMACS\
```

In base alla documentazione originale, nel caso della variabile di ambiente '**`EMACS`**' deve essere indicata la barra obliqua inversa finale.

A seconda della distribuzione di FreeDOS, può darsi che il file '`CONFIG.SYS`' debba essere sostituito con uno avente un nome differente. Potrebbe trattarsi del file '`FDCONFIG.SYS`'.

RxDOS

RxDOS è un altro progetto analogo a FreeDOS, scritto in maniera indipendente. È provvisto di un proprio interprete dei comandi e non ha ancora un suo sistema di installazione. Per provare il funzionamento di RxDOS ci si può avvalere solo di un dischetto, realizzato nel modo seguente:

1. si inizializza il dischetto in qualche modo, assicurando che alla fine sia disponibile un file system Dos-FAT;[2]
2. si esegue lo script '**`MAKEBOOT.BAT`**', il cui scopo è la predisposizione del settore di avvio nel dischetto;
3. si copiano ordinatamente nel dischetto i file elencati qui sotto.
   - '`RXDOSBIO.SYS`'
   - '`RXDOS.SYS`'
   - '`RXDOSCMD.EXE`'
   - '`RXDVDISK.SYS`'
   - '`AUTOEXEC.DEF`'
   - '`CONFIG.DEF`'

Riferimenti

- *FreeDOS*
  http://www.freedos.org

[1] **FreeDOS** GNU GPL

[2] Il dischetto non deve avere l'etichetta, ovvero non deve avere un nome.

Programmi di servizio vari 2057
Gnuplot 2057
Spreadsheet Calculator 2058
Ispell 2058
Perl 2058
Riferimenti 2058

Il progetto «GNUish» è una sorta di derivazione povera del progetto GNU, con lo scopo di rendere disponibile parte del software che compone il sistema GNU anche nei sistemi Dos. Il progetto ha un'importanza molto piccola, ma viene ancora mantenuto. Evidentemente, date le peculiarità dei sistemi Dos, il software che viene adattato non può avere le stesse potenzialità che ha invece in un sistema Unix.

I siti principali da cui si può ottenere copia del materiale prodotto dal progetto GNUish sono quelli elencati all'interno dal documento seguente:

- *http://www.math.utah.edu/docs/info/gnuish_1.html*

In questo capitolo viene mostrato il funzionamento di alcuni programmi, nell'ambito del sistema Dos, per i quali è il caso di spendere qualche parola.

Programmi di servizio vari

Molti dei programmi di servizio del progetto GNU sono disponibili anche per Dos. Tuttavia, è il caso di osservare alcune particolarità che possono confondere chi è abituato a usare sistemi Dos.

La prima cosa da notare è il fatto che i percorsi si possono indicare secondo lo stile Unix, utilizzando barre oblique normali. Per esempio:

```
C:\>MV C:/PRIMO/SECONDO C:/TERZO [ Invio ]
```

Diversamente, utilizzando lo stesso comando, ma secondo l'indicazione tipica del Dos, la cosa può funzionare ugualmente, oppure si possono presentare delle segnalazioni di errore. Bisogna tenere presente la possibilità.

Un'altra cosa da notare è l'uso dei caratteri jolly, che con questi programmi segue la logica di Unix, dove l'asterisco indica qualunque nome, senza trattare in modo speciale il punto di separazione dell'estensione:

```
C:\>CP C:/PRIMO/SECONDO/* C:/TERZO [ Invio ]
```

L'esempio mostra proprio questo fatto: vengono copiati tutti i file contenuti nella directory '`C:\PRIMO\SECONDO\`', nella directory '`C:\TERZO\`'.

Gnuplot

Per funzionare, questa edizione di Gnuplot richiede due file: '`GNUPLOT.EXE`' e '`GNUPLOT.GIH`'. Il primo dei due è l'eseguibile in grado di gestire la grafica VGA, mentre il secondo contiene le informazioni della guida interna.

Se si vuole accedere alla guida interna, è necessario che il file '`GNUPLOT.GIH`' si trovi nella directory corrente. Forse è sufficiente utilizzare il comando '**`APPEND`**' del Dos per risolvere il problema.

Spreadsheet Calculator

Il funzionamento generale di SC (Spreadsheet Calculator) è descritto nel capitolo u10. La versione per Dos funziona correttamente (è sufficiente disporre dell'eseguibile '**SC.EXE**'), riconoscendo anche l'uso dei tasti freccia, per cui non si è più costretti a utilizzare le lettere '**h**', '**j**', '**k**' e '**l**'.

Ispell

Questa edizione di Ispell richiede due file: 'ISPELL.EXE' e 'ISPELL.DIC'. Come si intuisce, il primo è l'eseguibile, mentre il secondo è il file del dizionario. Purtroppo, il file 'ISPELL.DIC' non è sostituibile o eliminabile; l'unica cosa che si può fare è predisporre un dizionario personalizzato che si richiama con l'opzione '**-p**'.

ISPELL [-d *dizionario_standard*] [-p *dizionario_personale*] *file*

Quella che si vede è la sintassi essenziale su cui si può contare nell'edizione di Ispell per Dos. Il file del dizionario standard, 'ISPELL.DIC', può essere collocato nella stessa directory in cui si trova il file eseguibile; altrimenti si deve usare l'opzione '**-d**' per indicarlo esplicitamente.

Il dizionario personale è un file di testo normale (Dos), che può anche essere creato inizialmente dallo stesso Ispell. L'esempio seguente, mostra il caso in cui si voglia analizzare il file 'LETTERA.TXT' attraverso il dizionario standard e il dizionario personale 'VOCAB.TXT'. Se il file 'VOCAB.TXT' non dovesse esistere, verrebbe creato per l'occasione.

```
C:\LETTERE>ISPELL -p VOCAB.TXT LETTERA.TXT [Invio]
```

Perl

L'edizione Dos dell'interprete Perl richiede due file: 'PERL.EXE' e 'PERLGLOB.EXE'. È sufficiente che questi siano disponibili nei percorsi degli eseguibili della variabile di ambiente '**PATH**'.

Bisogna tenere a mente che si tratta di una versione molto vecchia del linguaggio, per cui alcune novità non possono essere disponibili. Inoltre, l'avvio dei programmi può avvenire solo richiamando direttamente l'interprete:

```
C:\ESERCIZI>PERL FATT.PL 5 [Invio]
```

L'esempio mostra l'avvio del programma Perl contenuto nel file 'FATT.PL', che riceve un argomento costituito dal numero cinque.

Riferimenti

- François Pinard, *GNUish MSDOS Project*
  http://www.math.utah.edu/docs/info/MSDOS_1.html

The valuable DOS Freeware page

Introduction 2059
OS and GUI 2060
Utility 2060
Network 2061
Compilers 2062
Typesetting 2062
More Dos software sources 2062
Search engines 2063

Links to valuable free Dos programs working on low equipped computers.

This material appeared originally at 'http://www.geocities.com/SiliconValley/7737/', in 1996. Now it is incorporated inside the Italian document ''a2'', and it might be reached at the URI *http://a2.swlibero.org/the_valuable_dos_freeware_page.html* .
Questo materiale è apparso in origine, nel 1996, presso 'http://www.geocities.com/SiliconValley/7737/'. Adesso viene incorporato nel documento «a2» e può essere raggiunto attraverso l'URI *http://a2.swlibero.org/the_valuable_dos_freeware_page.html* . L'intento dell'autore è solo quello di continuare a curare un vecchio lavoro che potrebbe essere ancora utile, nonostante si tratti di riferimenti a software in parte libero e in parte solo gratuito, oltre che evidentemente obsoleto.

Introduction 2059
OS and GUI 2060
Utility 2060
Network 2061
Compilers 2062
Typesetting 2062
More Dos software sources 2062
Search engines 2063

Introduction

The Dos operating system meant much for many people. Today, proprietary Dos-like operating systems seem to be no more developed. In this situation, the only possible future for Dos is the ''free'' software, and it is not just a matter of money anymore.

Unfortunately, ''free'' is a word with many meanings. Today, this is still the biggest obstacle to the future of the Dos world. There is so much software for Dos, with so many different license agreements. The typical Dos user doesn't mind to it. But this problem prevents the realization of big serious projects based on it.

Today, the Dos world needs *philosophy*, and the GNU idea is still the right one (*http://www.gnu.org*).

The author of this space would like to list here only ''free software'' in the sense stated by the Free Software Foundation, **but it is impossible**, as there isn't enough good real free software for Dos.

The listed software is meant to work on i286 and below.

It is attempted to give some kind of classification about the legal condition of the software presented here. The definition used might be outdated, or there might be other wrong assumption. In particular, the definition ''public domain'' means here, in most cases, that there is the source, but there is no clear license statement.

Beside the URI links of some FTP services there is an additional ‘‘search link’’ that queries a FTP search engine for the same file. These additional links should be used when there are troubles with the main links.

Anyone can link this document anywhere, so, there is no need to ask for it. Anyway, it is better to link to this document using at the file name *http://a2.swlibero.org/the_valuable_dos_freeware_page.html* .

In the future, many links may disappear on this page, because of more selective choices concerning software license.

OS and GUI

- FreeDOS [1] *http://www.freedos.org*
- nanoDos, a FreeDOS distribution with some networking programs, ready to use/install: *ftp://na.mirror.garr.it/mirrors/AppuntiLinux/nanoDos/*

  Italian documentation can be found in chapter u62.
- FreeGEM [2] *http://l.webring.com/hub?ring=freegem*

Utility

archive, **backup**

- Gzip [3] - ‘`.GZ`’ archive compressor and extractor. *ftp://ftp.simtel.net/pub/simtelnet/msdos/compress/gzip124.zip* *http://www.google.com/search?q=gzip124.zip*
- TAR [4] - portable TAR - DOS/UNIX backup, compressor, with hardware support. *ftp://ftp.simtel.net/pub/simtelnet/msdos/arcers/tar320g.zip* *http://www.google.com/search?q=tar320g.zip*
- Untgz [5] - ‘`.TGZ`’, ‘`.TAR`’, ‘`.GZ`’, ‘`.ZIP`’ file extractor. *ftp://ftp.simtel.net/pub/simtelnet/msdos/arcers/untgz095.zip* *http://www.google.com/search?q=untgz095.zip*
- Info-ZIP [6] - ‘`.ZIP`’ compatible compression and extraction utility. *http://www.info-zip.org/pub/infozip/* *ftp://ftp.info-zip.org/pub/infozip/MSDOS/*
- Restaur [7] - Replacement for Dos Restore, *ftp://ftp.simtel.net/pub/simtelnet/msdos/diskutil/restaur1.zip* *http://www.google.com/search?q=restaur1.zip*

communication

- DosFax [8] - Send a fax using Dos command line *http://www.Adr.de/speicherplatz/cs/dosfax.htm*
- Bgfax [9] *http://www.blkbox.com/~bgfax/*
- Rifs [10] - Disk sharing over a serial line, *ftp://ftp.simtel.net/pub/simtelnet/msdos/lan/dosrifs2.zip* *http://www.google.com/search?q=dosrifs2.zip*

directory, **file**

- WCD [11] - Powerful chdir for Dos and Unix *http://www.xs4all.nl/~waterlan/*

disk

- Dug_ide [12] - Report ATA (IDE) disks geometry *ftp://ftp.simtel.net/pub/simtelnet/msdos/diskutil/dugide21.zip* *http://www.google.com/search?q=dugide21.zip*
- Fips [13] - Non-destructive splitting of hard disk partitions *ftp://ftp.simtel.net/pub/simtelnet/msdos/diskutil/fips15.zip* *http://www.google.com/search?q=fips15.zip*
- Part [14] - MBR partition manager *http://www.ranish.com/part/*

help

- NG_clone [15] - Norton Guides clone *ftp://ftp.simtel.net/pub/simtelnet/msdos/txtutl/ngclon11.zip* *http://www.google.com/search?q=ngclon11.zip*

shell

- DC [16] - The Dos Controller - A Norton Commander clone *ftp://ftp.simtel.net/pub/simtelnet/msdos/fileutil/dc-sk.zip* *http://www.google.com/search?q=dc-sk.zip*

system

- Cmos [17] - Save/Restore extended C/MOS *ftp://ftp.simtel.net/pub/simtelnet/msdos/sysutl/cmos93cd93.zip* *http://www.google.com/search?q=cmos93cd93.zip*
- Cmoser [18] - Save, restore and modify C/MOS memory *ftp://ftp.simtel.net/pub/simtelnet/msdos/sysutl/cmoser11.zip* *http://www.google.com/search?q=cmoser11.zip*
- Bios [19] - Save, restore, erase C/MOS memory; save a BIOS copy to a file *http://www.bockelkamp.de/software/discontinued/bios1351.zip*
- KGB [20] - Utility to monitor some Dos functions and reporting into a log file *ftp://ftp.simtel.net/pub/simtelnet/msdos/sysutl/kgb104.zip* *http://www.google.com/search?q=kgb104.zip*

text

- Vim [21] - VI improved, a small text editor that can handle very big files with low RAM *ftp://ftp.simtel.net/pub/simtelnet/msdos/editor/vim53d16.zip* *http://www.google.com/search?q=vim53d16.zip*

Network

packet driver

- PC/TCP Packet Driver Collection [22] *ftp://ftp.crynwr.com/drivers/* *ftp://ftp.crynwr.com/drivers/pktd11.zip*
- WATTCP [23] - TCP/IP library routines *http://www.wattcp.com/*
- DOS PPPD [24] - Dos port of Linux PPP packet driver *ftp://ftp.simtel.net/pub/simtelnet/msdos/pktdrvr/dosppp05.zip* *http://www.google.com/search?q=dosppp05.zip*
- Comring [25] - packet driver emulating ethernet over serial link(s) *http://wizard.ae.krakow.pl/~jb/ComRing/*

TCP/IP

- WATTCP apps [26] - some common client application using WATTCP library *http://www.smashco.com/wattcp/apps.zip*
- MiniTelnet [27] - TELNET client *http://www.smashco.com/wattcp/mt.zip*
- Bobcat [28] - Text based web browser *http://www.fdisk.com/doslynx/bobcat.htm* (derived from DosLynx, *ftp://ftp2.cc.ukans.edu/pub/WWW/DosLynx/*)
- Arachne (GPL) [29] - Graphical web browser *http://home.hetnet.nl/~ba8tian/arachne/arachne.htm* , *http://home.hetnet.nl/~ba8tian/arachne/175-gpl/ar175.htm* , *http://home.hetnet.nl/~ba8tian/arachne/175-gpl/a175gp75lf.zip*
- PCroute [30] - IP routing program for IBM PC *ftp://ftp.simtel.net/pub/simtelnet/msdos/network/pcrte224.zip* *http://www.google.com/search?q=pcrte224.zip*
- PPRD [31] - Turn a dedicated PC (XT/AT) into a LPD server *ftp://ftp.simtel.net/pub/simtelnet/msdos/lan/pprd200.zip* *http://www.google.com/search?q=pprd200.zip*
- NCSA Telnet [32] - Telnet, Ftp,... NCSA *http://archive.ncsa.uiuc.edu/SDG/Software/PCTelnet/* *ftp://ftp.ncsa.uiuc.edu/Telnet/DOS/* *ftp://ftp.simtel.net/pub/simtelnet/msdos/ncsatlnt/*
- NOS (KA9Q) [33] - A complete mini TCP/IP system *ftp://ftp.simtel.net/pub/simtelnet/msdos/tcpip/* *http://www.google.com/search?q=e920603.zip*

  To use NOS you need documentation, for example the package *ftp://ftp.simtel.net/pub/simtelnet/msdos/tcpip/intronos.zip*
- SSHDOS [34] - SSH client for Dos *http://sourceforge.net/projects/sshdos*

- Talk [35] - Talk client for Dos *http://www.smashco.com/wattcp/talk-13.zip*
- ABC-nslookup [36] - DNS query clients for Dos *http://www.smashco.com/wattcp/nslb01a.zip*

See also:

- Marc S. Ressl, *Dos Internet Pages* *http://www.fdisk.com/doslynx/*
- Smash-Co Communications, *TCP/IP for MS-DOS* *http://www.smashco.com/wattcp.asp*
- *The U-M Software Archive* *http://www.umich.edu/~archive/msdos/communications/wattcp/* *http://www.umich.edu/~archive/msdos/communications/packet/*

Compilers

assembler

See the FreeDOS project (*http://www.freedos.org*) for assembler compilers.

batch

- BAT2EXE [37] - Compile batch files for speed *ftp://ftp.simtel.net/pub/simtelnet/msdos/batchutl/bat2ex15.zip* *http://www.google.com/search?q=bat2ex15.zip*

C/C++

See the FreeDOS project (*http://www.freedos.org*) for C and C++ compilers.

Perl

Perl [38] - Practical Extraction Report Language *ftp://ftp.simtel.net/pub/simtelnet/msdos/perl/*

Rexx

- BREXX [39] - Rexx interpreter for Dos/Unix *http://ftp.gwdg.de/pub/languages/rexx/brexx/html/rx.html*

xBase

- nanoBase [40] - Mini, but nearly complete xBase *http://a2.swlibero.org/nanobase_1997.html*

See also:

- David Muir Shamoff, *Catalog of free compilers and interpreters* *http://www.idiom.com/free-compilers/*

Typesetting

- Nro [41] - A Nroff implementation for Dos *ftp://ftp.simtel.net/pub/simtelnet/msdos/txtutl/nroff1.zip* *http://www.google.com/search?q=nroff1.zip*
- Ghostscript [42] - ‘‘GNU’’ original edition - PostScript previewing, conversion, and printing *ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/gnu/*
- emTeX [43] - TeX-LaTeX distribution for Dos *ftp://www.ctan.org/tex-archive/systems/msdos/emtex/*

More Dos software sources

- *ftp://ftp.simtel.net/pub/simtelnet/msdos/*
- *http://garbo.uwasa.fi/pc/*

Search engines

- *http://www.shareware.com/*

[1] **FreeDOS** GNU GPL
[2] **FreeGEM** GNU GPL
[3] **Gzip** GNU GPL
[4] **TAR (Dos)** public domain
[5] **Untgz** GNU GPL
[6] **Info-ZIP** free software with special license
[7] **Restaur** cannot be sold for profit
[8] **DosFax** public domain
[9] **Bgfax** promised to become free software
[10] **Rifs** cannot be sold for profit
[11] **WCD** GNU GPL
[12] **Dug_ide** free of charge, with sources
[13] **Fips** GNU GPL
[14] **Part** public domain
[15] **NG_clone** public domain
[16] **DC** public domain (no license at all, and no sources)
[17] **Cmos** public domain
[18] **Cmoser** free of charge
[19] **Bios** free of charge
[20] **KGB** public domain
[21] **Vim** free software with special license
[22] **Crynwr packet driver collection** GNU GPL
[23] **WATTCP** free of charge library
[24] **DOS PPPD** mixed licenses
[25] **Comring** GNU GPL
[26] **WATTCP apps** cannot be sold
[27] **MiniTelnet** free software with a special license
[28] **Bobcat** GNU GPL
[29] **Arachne** GNU GPL
[30] **PCroute** cannot distribute modifications
[31] **PPRD** software non libero: licenza Artistic
[32] **NCSA Telnet** public domain
[33] **NOS** public domain
[34] **SSHDOS** GNU GPL
[35] **Talk** GNU GPL
[36] **ABC-nslookup** UCB BSD
[37] **BAT2EXE** public domain
[38] **Perl** GNU GPL or Artistic
[39] **BREXX** public domain
[40] **nanoBase** GNU GPL
[41] **Nro** public domain
[42] **Ghostscript** GNU GPL
[43] **emTeX** LPPL but some files have different conditions

Introduzione a ReactOS

Installazione 2065
Riferimenti 2066

ReactOS [1] è un progetto per la realizzazione di un sistema operativo conforme al funzionamento di MS-Windows, a partire da NT in su. Queste annotazioni sono state fatte a proposito della versione 0.1.2, dove il suo sviluppo non ha ancora portato a un sistema grafico funzionante e appare come un sistema Dos, anche se non è compatibile con gli eseguibili Dos standard. A partire dalle versioni 0.2.∗ è già disponibile un sistema grafico elementare.

Teoricamente ReactOS dovrebbe essere in grado di funzionare a partire da elaboratori i386 in su; tuttavia, è probabile che si riesca a utilizzare solo da i686 in su, escludendo anche i microprocessori compatibili.

Installazione

Allo stato della versione 0.1.2, il programma di installazione non è ancora in grado di gestire le partizioni e nemmeno di inizializzarle; pertanto, la partizione dove va collocato ReactOS deve essere preparata con altri strumenti (un sistema GNU/Linux o FreeDOS). Inizialmente conviene limitarsi all'utilizzo di un file system Dos-FAT16;.

Il programma di installazione tenta di riconoscere il settore di avvio del disco e della partizione in cui viene installato ReactOS; se questi settori risultano essere di qualche tipo particolare, non vengono modificati e poi il sistema non si avvia. Anche se la cosa è spiacevole, nel caso siano presenti altri sistemi nel disco fisso, può essere necessario fare in modo che nella partizione stabilita si avvii FreeDOS; in pratica, serve un dischetto di avvio di FreeDOS (capitolo u184), con il quale si deve ripristinare il settore di avvio del disco fisso:

```
A:>FDISK /MBR [Invio]
```

Quindi si deve trasferire il sistema minimo nella partizione, con il comando:

```
A:>SYS C: [Invio]
```

Naturalmente, la partizione in questione deve risultare attiva, ovvero deve essere quella che viene «avviata».

Per poter installare ReactOS, è necessario disporre di un CD-ROM, che si ottiene scaricando il file-immagine, seguendo le indicazioni contenute nel sito *http://reactos.com* . Il file-immagine dovrebbe risultare essere un file compresso, in formato '`.zip`'; una volta estratto, si può usare per incidere un CD-ROM.

Il CD che si ottiene dovrebbe essere autoavviabile, inserendosi automaticamente nelle procedura di installazione. L'installazione è relativamente breve, dal momento che il sistema è ancora allo stato iniziale del suo sviluppo. Al termine, se non ci sono stati problemi, viene sistemato anche il sistema di avvio; pertanto dovrebbe bastare riavviare l'elaboratore per mettere in funzione la copia di ReactOS appena installata.

ReactOS viene installato a partire dalla directory '`\ReactOS\`', ma il sistema di avvio rimane all'inizio ed è composto dai file '`\freeldr.sys`' e '`\freeldr.ini`'; eventualmente, il secondo di questi due può essere modificato per cambiare il menù di Avvio.

Una volta fatto funzionare il sistema, se è necessario, occorre intervenire nuovamente nel settore di avvio del disco fisso, in modo da poter usare sistemi come GRUB o LILO, attraverso i quali si deve fare in modo di avviare il primo settore della partizione contenente ReactOS, oltre che gli altri sistemi operativi che probabilmente sono installati nello stesso disco.

Riferimenti

- *ReactOS*
  http://reactos.com
- SourceForge.net: *ReactOS*
  http://sourceforge.net/projects/reactos

[1] **ReactOS** GNU GPL

DOSEMU: l'emulatore di hardware DOS compatibile

Predisporre un ambiente adatto al Dos all'interno di DOSEMU 2067
Un disco «C:» immagine 2067
Un disco «D:» virtuale o di rete 2068
La struttura essenziale del disco «D:» virtuale 2068
La configurazione di DOSEMU 2068
File «/etc/dosemu.users» 2068
File «/etc/dosemu.conf» 2069
Installare e utilizzare il Dos 2069
I dischi virtuali con «LREDIR.COM» 2071
Il mouse 2071
Un esempio di «AUTOEXEC.BAT» 2071
DOSEMU e le console virtuali 2071
DOSEMU da X 2071
DOSEMU e Mtools 2071
Implicazioni sulla gestione dei permessi 2072

DOSEMU [1] è fondamentalmente un emulatore dell'hardware x86 per vari sistemi Unix funzionanti su architettura i386. Il suo obiettivo è quello di permettere il funzionamento del sistema operativo Dos (MS-Dos o cloni). Si tratta di un progetto eternamente in fase di sviluppo (*alpha*), anche se da diversi anni è sufficientemente funzionante. Tuttavia non ci sono punti fermi: da una versione all'altra si possono incontrare novità imprevedibili.

Dal momento che l'emulazione riguarda l'hardware, il Dos deve essere installato all'interno di questo sistema di emulazione; quindi, è necessaria una copia di questo sistema operativo, insieme alla licenza d'uso.

DOSEMU permette di utilizzare la stessa copia installata del Dos su più terminali contemporaneamente. Se si intende concedere l'utilizzo simultaneo di una singola copia di questo sistema operativo, è necessario un numero maggiore di licenze d'uso, oppure una licenza multipla.

A fianco del lavoro su DOSEMU è anche in corso quello sul progetto FreeDOS per un sistema operativo Dos libero (capitolo u184).

Predisporre un ambiente adatto al Dos all'interno di DOSEMU

Perché il sistema operativo Dos possa funzionare all'interno di DOSEMU, occorre preparare un file-immagine di un disco Dos dal quale si possa effettuare l'avvio del Dos stesso. Questo file che viene descritto di seguito, viene visto dal Dos come disco '`C:`'.

Successivamente è conveniente predisporre uno spazio all'interno del file system del proprio sistema GNU/Linux da utilizzare per i programmi Dos che deve essere letto come un disco di rete.

Un disco «C:» immagine

Per effettuare l'avvio del Dos occorre che sia predisposta l'immagine di un disco di piccole dimensioni. Questo potrebbe essere un file contenuto nella directory '`/var/lib/dosemu/`', oppure '`/var/state/dosemu/`', il cui nome inizia normalmente per '`hdimage`'.

Attualmente, il file dovrebbe chiamarsi '`hdimage.first`' e al limite potrebbe essere un collegamento simbolico a un altro file che costituisce l'immagine vera e propria.

Se non esiste questo file è necessario copiarlo dal pacchetto sorgente. Il nome dovrebbe essere '`hdimage.dist`', o qualcosa di simile. Questa immagine deve essere preparata in seguito.

Un disco «D:» virtuale o di rete

In questa fase conviene preparare una directory che definisca l'inizio (la radice) del disco '`D:`' virtuale utilizzato dai programmi Dos. Stabiliamo che questo sia '`/var/emul/dos/`'. Da questo punto in poi, '`D:\`' è equivalente a '`/var/emul/dos/`'.

La struttura essenziale del disco «D:» virtuale

Il disco '`D:`' virtuale dovrebbe contenere alcune directory che riproducono in pratica il classico ambiente Dos:

- '`D:\TEMP\`'
  equivalente a '`/var/emul/dos/temp/`';
- '`D:\DOS\`'
  equivalente a '`/var/emul/dos/dos/`'.

Per evitare la proliferazione di directory temporanee, è possibile utilizzare al posto di '`/var/emul/dos/temp/`' un collegamento simbolico che punti a '`/tmp/`'.

```
# ln -s /tmp /var/emul/dos/temp [Invio]
```

La configurazione di DOSEMU

La configurazione di DOSEMU consiste nella modifica dei file '`/etc/dosemu.conf`' e di '`/etc/dosemu.users`'. Il file '`/etc/dosemu.users`' permette di definire gli utenti che possono utilizzare DOSEMU, mentre l'altro stabilisce tutte le altre caratteristiche.

Purtroppo, la configurazione di DOSEMU, specialmente per ciò che riguarda il file '`/etc/dosemu.conf`', è complessa e cambia da versione a versione. Inoltre, DOSEMU può costituire anche un problema per la sicurezza del sistema dal momento che di solito l'eseguibile '**`dos`**', deve essere SUID-root (cioè deve appartenere a '**`root`**' e avere il bit SUID attivato) per utilizzare funzionalità particolari dell'hardware (soprattutto l'adattatore grafico VGA).[2]

File «/etc/dosemu.users»

DOSEMU permette di distinguere alcune categorie di utenti, attribuendogli privilegi differenti, in base a una diversa configurazione nel file '`/etc/dosemu.conf`'. Tali categorie di utenti dipendono quindi dalla configurazione di questo file.

Il file '`/etc/dosemu.users`' può contenere righe di commento, introdotte dal simbolo '`#`', righe bianche o vuote, che vengono ignorate, e direttive espresse dalla sintassi seguente:

utente [***variabile_di_configurazione***...]

In pratica, si possono abbinare a un utente una o più variabili di configurazione che fanno riferimento a elementi del file '`/etc/dosemu.conf`'. È da osservare, in particolare, che si può indicare anche un utente particolare, '**`all`**', per fare riferimento a tutti gli utenti a cui non si fa menzione in modo esplicito.

A titolo di compromesso, viene mostrato un esempio di configurazione del file '`/etc/dosemu.users`' che dovrebbe essere sufficiente nella maggior parte delle situazioni. Si tratta in pratica della versione standard distribuita assieme a DOSEMU, con l'aggiunta di qualche utente ipotetico.


```
# This is a sample /etc/dosemu.users file
# For more details look at ./doc/README.conf

root c_all              # root is allowed to do all weird things
nobody guest            # variable 'guest' is checked in /etc/dosemu.conf
                        # to allow only DEXE execution
guest guest             # login guest treated as 'nobody'

# Utenti inseriti normalmente
tizio
caio
sempronio

# If you want to allow limited dosemu to all users, uncomment the line below
#all restricted         # all other users have normal user restrictions
```

Come si intuisce, l'utente '**`root`**' ha tutti i diritti necessari a compiere quello che vuole dall'interno di DOSEMU. Sono previsti gli utenti '**`nobody`**' e '**`guest`**', a cui sono concesse solo poche cose, mentre agli utenti '**`tizio`**', '**`caio`**' e '**`sempronio`**' sono concessi privilegi normali. Infine, appare commentata la direttiva '**`all restricted`**', con la quale si potrebbe consentire l'utilizzo di DOSEMU a tutti gli altri utenti, con privilegi ridotti.

File «/etc/dosemu.conf»

La preparazione di '`/etc/dosemu.conf`' è invece più delicata. Il file di esempio già fornito all'interno del pacchetto di distribuzione di DOSEMU è commentato molto dettagliatamente, però è anche molto complesso. Di seguito vengono indicate solo alcune parti particolarmente importanti. Le altre direttive di questo file, possono essere lasciate come sono, ignorandole, almeno fino a quando non si raggiunge una buona esperienza con l'uso di DOSEMU.

```
# Viene impostata la mappa della tastiera per uniformarsi alla
# disposizione dei tasti in Italia.
$_rawkeyboard = (1)     # bypass normal keyboard input, maybe dangerous
$_layout = "it"         # one of: finnish(-latin1), de(-latin1), be, it, us
                        # uk, dk(-latin1), keyb-no, no-latin1, dvorak, po
                        # sg(-latin1), fr(-latin1), sf(-latin1), es(-latin1)
                        # sw, hu(-latin2), hu-cwi, keyb-user
$_keybint = (on)        # emulate PCish keyboard interrupt


# Vengono definite le potenzialità dello schermo
# (per poter utilizzare la grafica, come impostato in questo
# esempio, occorre avviare il programma dos con i privilegi
# dell'utente root).
$_video = "vga"         # one of: plainvga, vga, ega, mda, mga, cga
$_console = (1)         # use 'console' video
$_graphics = (1)        # use the cards BIOS to set graphics
$_videoportaccess = (1) # allow videoportaccess when 'graphics' enabled
$_vbios_seg = (0xc000)  # set the address of your VBIOS (e.g. 0xe000)
$_vbios_size = (0x10000)# set the size of your BIOS (e.g. 0x8000)
$_vmemsize = (1024)     # size of regen buffer
$_chipset = ""          # one of: plainvga, trident, et4000, diamond, avance
                        # cirrus, matrox, wdvga, paradise
$_dualmon = (0)         # if you have one vga _plus_ one hgc (2 monitors)


# Viene definito l'uso dei dischetti e dell'immagine del disco C:.
$_floppy_a ="threeinch" # or "fiveinch" or empty, if not existing
$_floppy_b = ""         # ditto for B:

$_hdimage = "hdimage.first" # list of hdimages under /var/lib/dosemu
                        # assigned in this order such as
                        # "hdimage_c hdimage_d hdimage_e"
                        # If the name begins with '/dev/', then partition
                        # access is done instead of virtual hdimage such as
                        # "/dev/hda1" or "/dev/hda1:ro" for readonly
                        # Currently mounted devices and swap are refused.
                        # Hdimages and devices may be mixed such as
                        # "hdimage_c /dev/hda1 /dev/hda3:ro"
                        # Note: 'wholedisk' is _not_ supported.
$_hdimage_r = $_hdimage # hdimages for 'restricted access (if different)


# Viene definita la stampante.
$_printer = "lp"        # list of (/etc/printcap) printer names to appear as
                        # LPT1, LPT2, LPT3 (not all are needed, empty for none)
$_printer_timeout = (20)# idle time in seconds before spooling out

$_ports = ""            # list of portnumbers such as "0x1ce 0x1cf 0x238"
                        # or "0x1ce range 0x280,0x29f 310"
                        # or "range 0x1a0,(0x1a0+15)"
```

Installare e utilizzare il Dos

Il problema successivo è quello di riuscire a installare il Dos nel file-immagine che serve per effettuare l'avvio del Dos stesso. L'immagine in questione, che probabilmente è il file '`/var/lib/dosemu/hdimage.first`', contiene già una serie di programmi Dos che fanno parte di DOSEMU e come tali non vanno cancellati. Ma l'imma-

gine che viene distribuita così non è avviabile e il problema è proprio quello di inserirvi il kernel del Dos e l'interprete dei comandi '**COMMAND.COM**', salvo il caso in cui sia già presente una versione di FreeDOS.

1. **Preparazione di un dischetto di avvio**

   Per prima cosa occorre preparare un dischetto Dos avviabile che contenga qualche programma di servizio indispensabile. Da un elaboratore che stia eseguendo il sistema operativo Dos si procede come segue:

   ```
   C:> FORMAT A: /S [Invio]

   C:> COPY C:\DOS\SYS.* A: [Invio]

   C:> COPY C:\DOS\FDISK.* A: [Invio]
   ```

   Oltre a questi file converrebbe preparare nel dischetto un programma per la creazione e modifica di file di testo. Questo serve per preparare i file '`CONFIG.SYS`' e '`AUTOEXEC.BAT`'.

2. **Avvio del dischetto attraverso DOSEMU**

   È necessario quindi avviare il Dos contenuto nel dischetto appena creato attraverso DOSEMU. Per fare questo, dall'elaboratore GNU/Linux si avvia DOSEMU nel modo seguente:

   ```
   # dos -A [Invio]
   ```

   Se tutto è andato bene si avvia il Dos; quindi, dopo la richiesta della data e dell'ora appare l'invito classico (il *prompt*), per l'inserimento dei comandi attraverso la shell ('**COMMAND.COM**').

   ```
   A:\>
   ```

3. **Trasferimento del sistema**

   Per trasferire nel file-immagine il sistema contenuto nel dischetto, in modo da rendere questa immagine avviabile, occorre procedere prima con la creazione di un MBR (*Master boot record*):

   ```
   A:\> FDISK /MBR [Invio]
   ```

   quindi con il trasferimento del sistema:

   ```
   A:\> SYS C: [Invio]
   ```

   Se è andato tutto bene, adesso il disco '`C:`', cioè l'immagine, è pronto.[3]

4. **Controllo del disco C:**

   Il disco '`C:`' dovrebbe contenere alcuni file di DOSEMU. Per verificare il contenuto è sufficiente spostarsi in '`C:`'.

   ```
   A:\> C: [Invio]

   C:\> DIR [Invio]
   ```

5. **Modifica di config.sys**

   Trovandosi in '`C:`', potrebbe essere conveniente modificare i file '`CONFIG.SYS`' e '`AUTOEXEC.BAT`'. Si inizia con '`CONFIG.SYS`'.

   Si stabilisce di poter utilizzare tutte le lettere di unità (*drive*) a disposizione.

   ```
   LASTDRIVE=Z
   ```

   Si definisce attraverso il driver '`EMUFS.SYS`' di DOSEMU che la prossima lettera di disco a disposizione punti alla directory '`/var/emul/dos/`'. Di conseguenza, quella directory viene interpretata come disco '`D:`'

   ```
   DEVICE=C:\EMUFS.SYS /var/emul/dos
   ```

   Viene avviato il driver '`EMS.SYS`' di DOSEMU che si occupa della gestione della memoria estesa.

   ```
   DEVICE=C:\EMS.SYS
   ```

   Se in seguito se ne presenta l'opportunità, è sempre possibile apportare modifiche a questo file.

6. **Modifica di '`AUTOEXEC.BAT`'**

   Inizialmente il file non necessita di modifiche. È possibile vedere in seguito come configurare al meglio questo file.

7. **Conclusione dell'installazione**

   Per terminare la sessione di lavoro dell'installazione occorre fare terminare l'esecuzione di DOSEMU, avviato in precedenza con il comando '**dos -A**'. Per chiudere si utilizza il programma '**EXITEMU.COM**':

   ```
   C:\> C:\EXITEMU [Invio]
   ```

8. **Verifica**

   Se tutto è andato come previsto, il Dos è pronto. Si può provare ad avviare il Dos senza l'uso del dischetto semplicemente con il comando:

   ```
   $ dos [Invio]
   ```

   Se ha funzionato, si ottiene l'invito normale:

   ```
   C:\>
   ```

   Per uscire si utilizza il programma '**EXITEMU.COM**':

   ```
   C:\> EXITEMU [Invio]
   ```

I dischi virtuali con «LREDIR.COM»

Il programma '**LREDIR.COM**' è in grado di consentire l'accesso a porzioni del file system di GNU/Linux attribuendo una lettera di unità. Per esempio:

```
C:\> LREDIR X: \linux\fs\/home [Invio]
```

fa sì che il disco '`X:`' corrisponda al contenuto della directory '`/home/`'. Invece,

```
C:\> LREDIR Y: \linux\fs\${home} [Invio]
```

fa sì che il disco '`Y:`' corrisponda al contenuto della directory personale dell'utente che sta usando DOSEMU.

Il mouse

Teoricamente, DOSEMU è in grado di gestire da solo il mouse. In pratica potrebbe non essere così. In tal caso conviene provare ad avviare un programma apposito all'interno del '`CONFIG.SYS`' o di '`AUTOEXEC.BAT`'.

Un esempio di «AUTOEXEC.BAT»

Nell'esempio seguente viene utilizzato un programma per la gestione del mouse estraneo a DOSEMU. Il disco '`D:`' è stato definito implicitamente all'interno di '`CONFIG.SYS`' attraverso '**DEVICE=C:\EMUFS.SYS /var/emul/dos**'.

```
@echo off

LREDIR H: linux\fs\${home}
LREDIR R: linux\fs\/mnt/cdrom

PROMPT=$p$g
PATH=c:\
PATH=%PATH%;D:\;D:\DOS

SET TEMP=D:\TEMP

D:

D:\DOS\MOUSE

ECHO "Questo è DOSEMU. Benvenuto!"
```

DOSEMU e le console virtuali

Quando viene avviato il Dos attraverso DOSEMU, questo opera nella console virtuale sulla quale ci si trova. Di solito, per passare da una console virtuale all'altra è sufficiente premere la combinazione [*Alt F1*] o [*Alt F2*]... Quando ci si trova su una console virtuale all'interno della quale sta funzionando il Dos, per passare a un'altra si agisce con la combinazione [*Ctrl Alt F1*] o [*Ctrl Alt F2*]...

DOSEMU da X

Per avviare il Dos in una finestra del sistema grafico X, conviene avviare DOSEMU attraverso '**xdos**' che normalmente è un collegamento simbolico a '**dos**'.

Nelle sezioni precedenti si è visto l'uso del file-immagine '`/var/lib/dosemu/hdimage`', che costituisce normalmente il disco '`C:`' per DOSEMU. Questo file non è gestibile con strumenti Unix normali, soprattutto perché non è un'immagine standard. Si tratta dell'immagine di un piccolo disco fisso contenente una partizione, con l'aggiunta di un'intestazione aggiuntiva.

Questo disco '`C:`' può essere utilizzato principalmente attraverso strumenti Dos all'interno di DOSEMU, così come è stato già mostrato, oppure può essere raggiunto anche tramite Mtools, purché configurato opportunamente. Infatti, è sufficiente informare Mtools sulla posizione esatta in cui ha inizio la prima partizione all'interno del file-immagine, per potervici accedere anche con questo strumento. Potrebbe trattarsi della direttiva seguente, nel file di configurazione '`/etc/mtools.conf`'.

```
drive n: file="/var/lib/dosemu/hdimage.first" partition=1 offset=128
```

In tal modo, per Mtools, il disco '`N:`' corrisponderebbe al disco '`C:`' di DOSEMU.

È importante fare attenzione al valore dello scostamento (*offset*) che potrebbe cambiare da una versione all'altra di DOSEMU.

Implicazioni sulla gestione dei permessi

Il Dos non è un sistema operativo multiutente e di conseguenza non è in grado di attribuire dei permessi ai file. Quando si utilizza il Dos all'interno di DOSEMU, i permessi vengono gestiti in modo predefinito.

Quando si crea un file gli vengono attribuiti i permessi predefiniti in base a quanto stabilito con la maschera dei permessi; inoltre, l'utente e il gruppo proprietario corrispondono all'utente che ha avviato DOSEMU e al gruppo cui questo utente appartiene.

Quando si accede a un file, l'apparenza delle caratteristiche di questo cambiano a seconda che l'accesso avvenga da parte di un utente rispetto a un altro: l'utente che ha creato il file può modificarlo, un altro potrebbe trovarlo protetto in sola lettura.

In particolare, i file contenuti nel file-immagine che costituisce il disco '`C:`' hanno le proprietà e i permessi del file-immagine stesso.

Ma il Dos non è in grado di gestire tutte le finezze che può invece amministrare un sistema Unix, di conseguenza, quando si tenta di fare qualcosa che i permessi non consentono, si ottengono per lo più delle segnalazioni di errore che normalmente non si vedono quando si usa il Dos da solo senza emulazioni.

Quando si utilizza il Dos con DOSEMU su un sistema al quale accede un solo utente, non dovrebbero porsi problemi: basta che l'unico utente utilizzi sempre lo stesso nominativo (lo stesso UID). Quando lo si utilizza invece in un sistema al quale accedono più utenti, è ragionevole desiderare che i dati personali possano essere inaccessibili agli altri; quindi, questo modo trasparente di gestire i permessi può essere solo positivo. Quando si vogliono gestire alcune attività in gruppo si può aggirare eventualmente l'ostacolo utilizzando un utente comune creato appositamente per quel compito.

Un'ultima annotazione deve essere fatta per i file eseguibili che non necessitano dei permessi di esecuzione, come invece richiederebbe GNU/Linux. È generalmente sufficiente che ci siano i permessi di lettura. A volte sono necessari anche quelli in scrittura, ma prima di dare questi permessi è meglio verificare, onde evitare di lasciare campo libero a un possibile virus.

[1] **DOSEMU** GNU GPL

[2] Se ci si accontenta di uno schermo a caratteri, senza grafica e senza cornici, non dovrebbe essere necessario attivare il bit SUID.

[3] Il comando '**`FDISK /MBR`**' riguarda precisamente MS-Dos, mentre nel caso di cloni le cose potrebbero essere differenti; per esempio potrebbe essere necessario avviare il programma nel modo solito e poi specificare la richiesta selezionando una voce da un menù.

Parte xli

Sistemi operativi alternativi

«

Sistemi operativi alternativi di un certo rilievo 2077

Riferimenti 2078

Syllable: introduzione 2079

Installazione 2079

Utenze 2082

Installazione dei pacchetti applicativi 2082

Riferimenti 2083

Syllable: utilizzo sommario 2085

Configurazione 2085

Cattura dello schermo 2086

Riferimenti 2087

Plan 9: installazione 2089

Dischetto di installazione e preparazione della distribuzione 2089

Installazione nel disco fisso 2092

Avvio e arresto del sistema installato 2097

Riferimenti 2097

UNIX di ricerca 2099

Caratteristiche dei primi sistemi UNIX e BSD 2100

SIMH e il PDP-11 2102

2.11BSD 2109

Installazione di file-immagine pronti 2118

Derivazioni di UNIX per hardware ridotto 2129

Programmi di servizio 2131

Riferimenti 2132

Riferimenti 2078

Oltre ai soliti sistemi GNU (GNU/Linux, GNU/Hurd, ecc.) e *BSD, esiste lo sviluppo o il progetto di altri sistemi operativi, più o meno liberi, che possono rivelarsi di un certo interesse. In questo capitolo viene fatto l'elenco di alcuni di questi sistemi operativi alternativi, la cui licenza rientra almeno nella categoria di quelle approvate da OSI, Open Source Initiative, *http://www.opensource.org* .

Si tenga presente il fatto che le informazioni annotate qui possono essere poco accurate, soprattutto in considerazione degli sviluppi che i vari progetti possono prendere.

Progetto	Licenza	Siti	Annotazioni
AROS	APL	*http://www.aros.org*	*Amiga research operating system* è un progetto per lo sviluppo di un sistema operativo compatibile con Amiga, inizialmente per elaboratori i386.
OpenDarwin	APSL (Apple public source license)	*http://www.opendarwin.org*	Una delle varianti BSD.
GNU-Darwin	APSL (Apple public source license), GNU GPL e altre	*http://www.gnu-darwin.org*	Variante GNU di OpenDarwin.
ELKS	GNU GPL	*http://elks.sourceforge.net/* *http://sourceforge.net/projects/elks/*	*Embeddable linux kernel subset* è un progetto che intende realizzare un sistema operativo per elaboratori con architettura i86 (a 16 bit), a partire da un sottoinsieme di funzionalità di GNU/Linux. ELKS è descritto nel capitolo u177.
FreeDOS	GNU GPL	*http://www.freedos.org*	È un progetto per la realizzazione di un sistema operativo libero compatibile con il Dos, anche sull'architettura i86 (a 16 bit). FreeDOS è descritto anche nel capitolo u184.
Haiku	MIT	*http://www.haiku-os.org/*	Il progetto, che originariamente aveva il nome OpenBeOS, punta alla realizzazione di un sistema operativo compatibile con BeOS.
Minix	simile a BSD	*http://www.cs.vu.nl/~ast/minix.html* *http://www.cs.vu.nl/pub/minix/*	Si tratta di un sistema Unix, nato per motivi didattici, in grado di funzionare anche su elaboratori con architettura i86 (a 16 bit). Minix è descritto nel capitolo u176.
Plan 9	Lucent Public License	*http://plan9.bell-labs.com/plan9dist/index.html*	È un sistema operativo innovativo, inteso come il successore di UNIX. È descritto a partire dal capitolo u192.

Progetto	Licenza	Siti	Annotazioni
ReactOS	GNU GPL	*http://reactos.com* *http://sourceforge.net/projects/reactos*	È un progetto per la realizzazione di un sistema operativo conforme al funzionamento di MS-Windows, a partire da NT in su. ReactOS è descritto nel capitolo u187.
Syllable	GNU GPL	*http://syllable.sourceforge.net* *http://sourceforge.net/projects/syllable*	Si tratta di un progetto derivato da AtheOS (di Kurt Skauen), per un sistema operativo compatibile con le specifiche POSIX, ma con delle particolarità rispetto ai sistemi Unix tradizionali. È descritto nei capitoli u190 e u191.

Riferimenti

- *OSDev ring*
  http://www.osdev.org/

Syllable: introduzione

Installazione 2079
Utenze 2082
Installazione dei pacchetti applicativi 2082
Riferimenti 2083

Syllable [1] è un sistema operativo orientato verso lo standard POSIX, distaccandosi leggermente dalle convenzioni comuni di un sistema GNU o *BSD. La sua caratteristica più appariscente è quella di funzionare in modo esclusivamente grafico, anche se poi si usano le solite finestre di terminale.

Syllable nasce come derivazione di AtheOS, [2] iniziato da Kurt Skauen, che attualmente sembra non essere più sviluppato attivamente.

In generale, Syllable (e il predecessore AtheOS) si può considerare un sistema operativo completo, con alcune limitazioni sulle componenti hardware che si possono usare effettivamente. Normalmente, un elaboratore con microprocessore i586, o compatibile, mouse PS/2 o seriale (se si tratta di un mouse seriale questo va collegato nella prima porta seriale), grafica VESA, dovrebbe funzionare regolarmente. Qualche problema, forse, si può incontrare con le schede di rete, nel senso che i modelli compatibili sono pochi.

Installazione

Nelle prime edizioni di Syllable ci sono difficoltà ad accedere ai lettori CD-ROM, pertanto viene mostrato un metodo di installazione molto spartano, compatibile anche con AtheOS nella versione 0.3.7.

In pratica è necessario disporre in qualche modo di un'altra partizione, in formato Dos-VFAT, che contenga l'archivio compresso da estrarre durante l'installazione. Pertanto, inizialmente bisogna agire usando altri strumenti, come un mini sistema GNU/Linux.

Da quanto appena esposto, si può comprendere che servono due partizioni primarie, la prima di tipo Dos-VFAT, inizializzata correttamente, e la seconda di un tipo non meglio precisato, dal momento che non sembra essere stato concordato un codice particolare per le partizioni contenenti un file system Syllable.

Una volta prelevato l'archivio compresso contenente la copia del file system di partenza, questo va copiato nella partizione con file system Dos-VFAT. Fatto questo ci si può occupare della preparazione dei dischetti di avvio, che potrebbero essere tre o più. Naturalmente vanno prelevati i file-immagine di questi dischetti, da riprodurre secondo le solite modalità, per esempio nel modo seguente se si usa un sistema GNU/Linux, avendo cura naturalmente di sostituire ogni volta il dischetto:

```
# cp syllable*1.img /dev/fd0 [Invio]

# cp syllable*2.img /dev/fd0 [Invio]

# cp syllable*3.img /dev/fd0 [Invio]
```

Disponendo dei dischetti si può inserire il primo nell'elaboratore in cui si vuole installare Syllable, riavviandolo. Viene richiesto di sostituire il primo con il secondo e poi con il terzo. A un certo punto appare la grafica e una finestra di terminale, con il quale vanno impartiti manualmente i comandi necessari.

Appena si dispone del terminale, si procede inizializzando la partizione che deve ospitare il sistema operativo; quindi vanno innestate entrambe per trasferire i dati. In questi esempi viene mostrato un invito fittizio, che potrebbe essere diverso dal reale, secondo le convenzioni comuni usate a proposito dei sistemi Unix.

Si suppone che la partizione che deve ospitare il file system di Syllable sia la seconda in assoluto.

Syllable usa una sua struttura particolare per i file di dispositivo (creati dinamicamente dal kernel) e per quanto riguarda l'accesso

a una partizione di un disco ATA, si può usare una delle notazioni seguenti:

```
/dev/disk/bios/hdx/n
```

```
/dev/disk/ide/hdx/n
```

La differenza tra i due tipi di notazione sta nella sigla '**bios**' o '**ide**', che identifica il tipo di gestione software utilizzato per accedere ai dischi ('**ide**' fa riferimento al bus «ATA»). I file di dispositivo di Syllable (così come per AtheOS) sono virtuali e creati dinamicamente dal sistema; pertanto, la scelta dipende anche dal fatto che il file di dispositivo sia disponibile effettivamente. In generale, il primo tipo dovrebbe essere disponibile quando si usano i dischetti di installazione (si veda anche la tabella u190.3).

Si osservi che la sigla ***x*** va sostituita con una lettera alfabetica minuscola, corrispondente alla posizione del disco ATA (il primo disco corrisponde a '**hda**'); inoltre, il numero finale rappresenta la partizione (primaria o estesa che sia), a partire da zero. Pertanto, per fare riferimento alla seconda partizione del primo disco si può usare il nome '`/dev/disk/bios/hda/1`'.

```
# format /dev/disk/bios/hda/1 afs Pippo [Invio]
```

Con il comando appena mostrato si inizializza la partizione usando un file system AFS (il formato nativo di Syllable e di AtheOS), a cui viene anche attribuito un nome obbligatorio (in questo caso è «Pippo»).

Il programma '**format**' non si accorge se la partizione non è quella che dovrebbe essere, pertanto se si va a inizializzare la partizione sbagliata si perdono i dati che questa contiene.

Una volta che l'inizializzazione è stata completata, si innestano le due partizioni, creando prima le directory necessarie all'innesto. Si suppone che la partizione contenente l'archivio da estrarre sia la prima:

```
# mkdir /dos [Invio]

# mount /dev/disk/bios/hda/0 /dos [Invio]

# mkdir /afs [Invio]

# mount /dev/disk/bios/hda/1 /afs [Invio]
```

Come si può intuire, il tipo di file system viene determinato automaticamente.

Si osservi che per motivi di spazio, i dischetti di avvio usati per installare Syllable potrebbero non avere certi comandi. Per esempio, potrebbe mancare '**unmount**' (contrariamente alla tradizione Unix è questo il nome del comando da usare per eseguire il distacco di un disco), quindi occorre fare queste operazioni con cura.

Si passa così a estrarre l'archivio contenente la copia del sistema:

```
# cd /afs [Invio]

# tar xzpvf /dos/base-syllable*.tgz [Invio]
```

Si può osservare che vengono prodotte solo le directory '`atheos/`' e '`boot/`', perché tutto il resto della struttura viene creato in modo virtuale durante il funzionamento del sistema operativo.

Per la precisione, durante il funzionamento queste directory appaiono discendere da '`/boot/`'; pertanto si possono vedere come '`/boot/atheos/`' e '`/boot/boot/`'.

Syllable (come anche AtheOS) utilizza GRUB per l'avvio del sistema, pertanto occorre verificare che il file '`boot/grub/menu.lst`' sia configurato correttamente. L'esempio seguente si riferisce proprio all'uso della seconda partizione:

```
timeout 5

title  Syllable
root   (hd0,1)
kernel /atheos/sys/kernel.so root=/dev/disk/ide/hda/1
module /atheos/sys/drivers/fs/afs
module /atheos/sys/drivers/dev/disk/ata
```

Si comprende intuitivamente che potrebbe essere necessario modificare le direttive '**root**' e '**kernel**', se la partizione fosse un'altra.

Nel caso fosse necessario modificare questo file, si deve avviare un programma per la modifica dei file di testo. I dischetti di installazione mettono a disposizione il programma Aedit, corrispondente all'eseguibile '**aedit**' (nel caso di AtheOS si tratta invece di Jed, associato all'eseguibile '**jed**'). Il programma in questione dovrebbe risultare molto intuitivo per il suo utilizzo.

Si osservi comunque che i moduli indicati rappresentano il minimo indispensabile per avviare il sistema, ma se si intende accedere al file system contenuto in un dischetto o in un CD-ROM, conviene aggiungerne altri. Si suggerisce un'impostazione iniziale come quella seguente, da estendere in seguito in base alle esigenze particolari:

```
timeout 5

title  Syllable
root   (hd0,1)
kernel /atheos/sys/kernel.so root=/dev/disk/ide/hda/1
module /atheos/sys/drivers/fs/afs
module /atheos/sys/drivers/fs/iso9660
module /atheos/sys/drivers/fs/ext2
module /atheos/sys/drivers/fs/fatfs
module /atheos/sys/drivers/dev/disk/ata
module /atheos/sys/drivers/dev/disk/bios
```

Per installare il settore di avvio di GRUB occorre riavviare il sistema, dopo aver reinserito il primo dei dischetti dell'installazione. Per ottenere un riavvio corretto è sufficiente premere la combinazione di tasti [*Ctrl Alt Canc*]. Quando appare il menù di GRUB del dischetto, basta premere [*Esc*] per impedire l'avvio e poi [*c*] per inserire a mano i comandi di GRUB:

```
grub> root (hd0,1) [Invio]

grub> setup (hd0) [Invio]
```

Il secondo di questi due comandi può riferirsi eventualmente a '**(hd0,1)**', cioè all'inizio della partizione invece che all'inizio del disco intero, se si intende gestire all'esterno di Syllable l'avvio di altri sistemi operativi che convivono nello stesso disco:

```
grub> root (hd0,1) [Invio]

grub> setup (hd0,1) [Invio]
```

Fatto questo (scegliendo, a ragion veduta, uno dei due modi proposti), si può provare a riavviare l'elaboratore senza più il dischetto di avvio.

Tabella u190.3. Alcuni esempi di file di dispositivo di Syllable.

Percorso del file di dispositivo	Descrizione
`/dev/disk/bios/hdx/raw` `/dev/disk/ide/hdx/raw`	Fa riferimento a tutto il disco fisso corrispondente alla lettera ***x***, a cui si accede attraverso le funzioni del BIOS, oppure attraverso una modalità più evoluta.
`/dev/disk/bios/hdx/n` `/dev/disk/ide/hdx/n`	Fa riferimento alla partizione ***n*** (primaria o estesa che sia) del disco fisso corrispondente alla lettera ***x***, a cui si accede attraverso le funzioni del BIOS, oppure attraverso una modalità più evoluta.
`/dev/disk/ide/cdx/raw`	Fa riferimento a tutto il CD-ROM collocato nel lettore corrispondente alla lettera ***x***.
`/dev/disk/bios/fdx/raw`	Fa riferimento a tutto il dischetto collocato nell'unità corrispondente alla lettera ***x***.
`/dev/misc/com/n`	Fa riferimento alla porta seriale ***n***.
`/dev/misc/ps2aux`	Fa riferimento al mouse PS/2.

Utenze

Una volta riavviato il sistema senza più i dischetti, appare una richiesta di identificazione, secondo il modo tradizionale dei sistemi Unix, ma di tipo grafico. Inizialmente sono disponibili due utenti: '**root**', associato alla parola d'ordine '**root**', e '**guest**', associato alla parola d'ordine '**guest**'. Naturalmente il primo utente è l'amministratore, mentre il secondo rappresenta il solito utente comune.

La creazione di utenze nuove può avvenire attraverso il programma '**Users**' (si osservi l'iniziale maiuscola), accessibile dal menù *System*, alla voce *Users*. Il programma guida in pratica alla compilazione del file '`/etc/passwd`', ma manca la possibilità di creare dei gruppi, pertanto per questo occorre intervenire in modo manuale.

```
# Users [Invio]
```

Nel file '`/etc/passwd`', il campo della parola d'ordine cifrata è ottenuto attraverso l'algoritmo MD5 e nelle prime versioni di Syllable (come di AtheOS) occorreva generarla attraverso il programma '**crypt**':

```
# crypt evviva [Invio]

  Password: '$1$$Pu4nMUbz1apLvoCuSUX9e.'
```

L'esempio mostra in che modo si potrebbe ottenere la stringa cifrata corrispondente alla parola d'ordine «evviva».

Si suppone di avere creato l'utente '**tizio**' e il gruppo '**tizio**' come dagli estratti seguenti che si riferiscono rispettivamente ai file '`/etc/passwd`' e '`/etc/group`':

```
tizio:$1$$Pu4nMUbz1apLvoCuSUX9e.:1000:1000:Tizio Tizi:/home/tizio:/bin/bash
```

```
tizio:*:1000:
```

A questo punto è necessario predisporre anche la directory personale di questo utente '**tizio**', partendo convenientemente da una copia di quella dell'utente '**guest**', sistemando la proprietà di file e directory:

```
# cd /home [Invio]

# cp -Rv guest tizio [Invio]

# chown -R tizio:  tizio [Invio]
```

Si osservi che tutte queste modifiche sono avvenute facendo riferimento alle collocazioni convenzionali di un sistema Unix, ma in pratica, directory come '`/home/`' e '`/etc/`', sono in realtà dei collegamenti simbolici che puntano a una struttura contenuta all'interno di '`/boot/atheos/`'. Come già accennato, questi collegamenti, così come la directory '`/dev/`', scompaiono dal file system quando il sistema operativo si arresta. In pratica, è come dire che il file system principale viene creato nella memoria centrale e su di esso, precisamente nella directory '`/boot/`', viene innestato il file system contenuto nel disco.

Installazione dei pacchetti applicativi

Syllable (come anche AtheOS) ha un proprio modo di gestire le applicazioni, che dovrebbe consentire un'installazione e una rimozione relativamente semplici, senza l'uso di un sistema di tracciamento come avviene nei sistemi GNU comuni. In pratica, un programma realizzato appositamente per Syllable dovrebbe poter essere copiato così come si trova (con la sua struttura di sottodirectory), in qualunque punto del file system, per risultare funzionante. Ovviamente ci può essere il problema del percorso di avvio degli eseguibili, ma a parte questo, il programma dovrebbe essere in grado di trovare tutto quello che lo riguarda senza altri problemi.

Naturalmente, questo tipo di idea è buona, ma non va d'accordo con le convenzioni dei programmi realizzati per i sistemi Unix comuni, che invece richiedono di essere installati secondo la gerarchia di directory tradizionale, mescolando tra loro i vari file.

Per compensare questo problema, quando si installano pacchetti applicativi che sono stati adattati per Syllable, occorre seguire una procedura particolare. Supponendo di voler installare il pacchetto contenuto nell'archivio '`mc-4.1.bin.1.tgz`', che si trova nella directory '`/tmp/`', si procede con l'estrazione a partire dalla directory '`/usr/`':

```
# cd /usr [Invio]

# tar xzpvf /tmp/mc-4.1.bin.1.tgz [Invio]
```

L'estrazione del pacchetto crea una sottodirectory a partire da '**`/usr/`**', che probabilmente ha lo stesso nome del pacchetto ('`mc/`' in questo caso). Osservato questo si usa il programma '**pkgmanager**', con l'opzione '**-a**' e il percorso della directory che riguarda il pacchetto:

```
# pkgmanager -a /usr/mc [Invio]
```

Volendo invece disinstallare un pacchetto del genere, si elimina prima la sua directory, quindi si usa '**pkgmanager**' con l'opzione '**-r**', per togliere tutti i riferimenti che prima lo rendevano compatibile con il sistema di Syllable:

```
# rm -r /usr/mc [Invio]

# pkgmanager -r /usr/mc [Invio]
```

Riferimenti

- *Syllable*
  http://syllable.sourceforge.net
  http://sourceforge.net/projects/syllable
- Kurt Skauen, *AtheOS*
  http://sourceforge.net/projects/atheos

[1] **Syllable (kernel)** GNU GPL

[2] **AtheOS (kernel)** GNU GPL

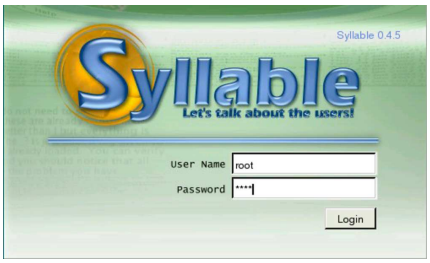
Syllable: utilizzo sommario

Configurazione 2085
Cattura dello schermo 2086
Riferimenti 2087

Syllable è un sistema operativo che dipende dalla grafica e si è quasi costretti a usarla, anche per la configurazione del suo funzionamento.

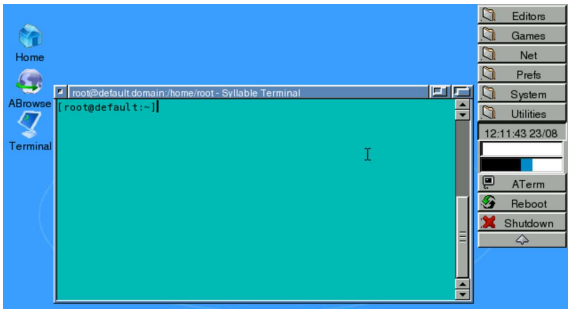
Si osservi che la grafica di Syllable non viene gestita da X come avviene di solito nei sistemi GNU, ma da un sistema scritto appositamente.

Figure u191.1. Identificazione dell'utente con Syllable.

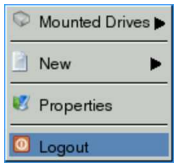


In condizioni normali, dopo l'identificazione dell'utente, appare una sorta di menù grafico, da dove si accede a funzionalità importanti. In particolare, se l'utente che accede è l'amministratore ('**`root`**'), questo menù contenere anche le voci che consentono di fermare o di riavviare il sistema.

Figure u191.2. Syllable con il menù e un terminale grafico, utilizzato dall'utente '**`root`**'.



Se con il mouse si fa clic con il tasto destro, quando il puntatore è su una superficie grafica libera, si ottiene un menù a scomparsa, che in particolare consentirebbe di chiudere la sessione di lavoro:

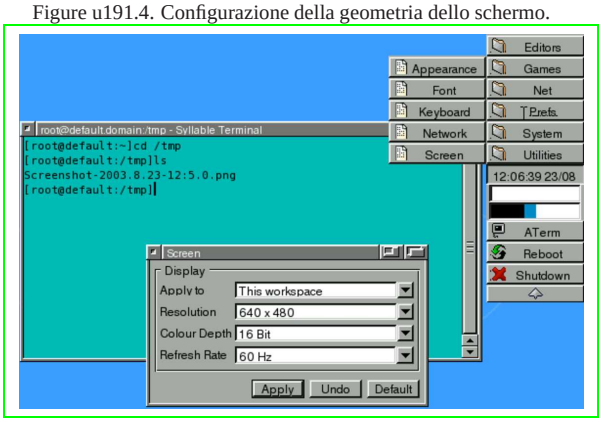


Configurazione

La configurazione di funzionalità importanti è riservata all'utente '**`root`**'. Di solito anche gli utenti comuni possono accedere, ma senza poter attuare delle modifiche.

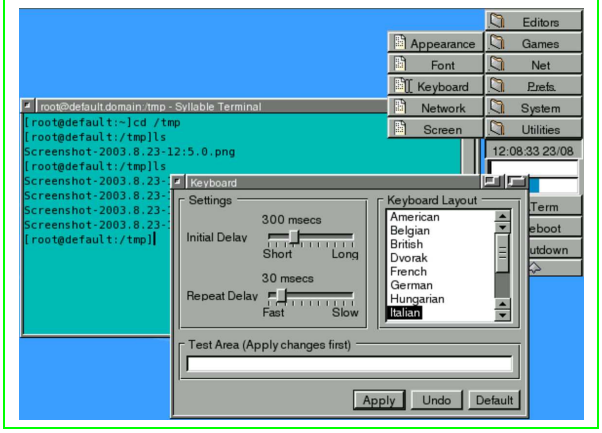
La configurazione dello schermo, per ciò che riguarda aspetti importanti come la risoluzione e la profondità di colori, può essere modificata selezionando la voce *Screen* del menù *Prefs*, oppure richiamando il programma '**`Prefs-Screen`**'.

Figure u191.4. Configurazione della geometria dello schermo.



Si accede alla configurazione della tastiera selezionando la voce *Keyboard* del menù *Prefs*, oppure richiamando il programma '**`Prefs-Keyboard`**'.

Figure u191.5. Configurazione della tastiera.



La gestione delle utenze, o più precisamente del file '`/etc/passwd`', si ottiene selezionando la voce *Users* del menù *System*, oppure richiamando il programma '**`Users`**'.

Figure u191.6. Gestione delle utenze.

Gli utenti comuni possono accedere a questo programma per modificare la propria parola d'ordine.

Cattura dello schermo

La cattura dell'immagine dello schermo si ottiene semplicemente premendo il tasto [*Stampa*] ([*Print Screen*] nelle tastiere per la lingua inglese). Questo fatto genera un file nella directory '`/tmp/`', con un nome simile al modello seguente:

`Screenshot-`***data*****-***orario***`.png`**

Naturalmente, se servono, questi file vanno trasferiti altrove, perché a ogni riavvio tutto verrebbe perduto.

Riferimenti

- *AtheDocs*
  originariamente presso: *http://www.other-space.com/athedocs*

Plan 9: installazione

Dischetto di installazione e preparazione della distribuzione 2089
Installazione nel disco fisso 2092
Avvio e arresto del sistema installato 2097
Riferimenti 2097

Plan 9 [1] è un sistema operativo inteso come il successore di UNIX. Attualmente è distribuito a partire dall'indirizzo *http://plan9.bell-labs.com/plan9dist/index.html* .

Plan 9 è disponibile inizialmente per l'architetture i386, con una compatibilità limitata a un insieme ristretto di componenti hardware. In generale, un elaboratore i386 relativamente recente, con un bus PCI, tastiera e mouse PS/2, dovrebbe funzionare correttamente.

Questo capitolo descrive brevemente la procedura per l'acquisizione del software e la sua installazione.

In questo capitolo si mostra l'installazione con la predisposizione di un file system Kfs, che è considerato superato. Il motivo di questa preferenza sta nel fatto che inizialmente la documentazione disponibile fa riferimento in modo prevalente al file system Kfs e può risultare troppo difficile, allo stato attuale, la gestione di un tipo più sofisticato.

Dischetto di installazione e preparazione della distribuzione

Attualmente, la distribuzione di Plan 9 avviene soltanto a partire dal sito citato all'inizio del capitolo. Per iniziare è necessario disporre di un dischetto di avvio, preparato a partire da un file-immagine ottenuto da quel sito; tuttavia, questa immagine non è unica, dal momento che c'è un file di configurazione da predisporre e probabilmente ci può essere la necessità di selezionare un kernel adeguato alle caratteristiche dell'elaboratore. Questo file-immagine viene così confezionato in base alle specifiche indicate, inoltre il dischetto contiene una sorta di numero di serie a cui è associato un tempo di scadenza, dopo il quale, se non è ancora stata completata l'installazione, bisogna prelevare un'altra copia di questo file-immagine con un nuovo numero di serie.

Prima di ottenere il file-immagine del dischetto viene richiesto di approvare la licenza; quindi si passa a un modulo da compilare con le caratteristiche salienti dell'elaboratore nel quale si vuole installare Plan 9.

Una volta ottenuto il file-immagine del dischetto, lo si trasferisce facilmente in un dischetto da 1440 Kibyte; per esempio così se si dispone di un sistema GNU/Linux:

```
# cp 9disk.flp /dev/fd0 [Invio]
```

Come si può intendere, '`9disk.flp`' è il file-immagine ottenuto.

Una volta prelevato il file-immagine, viene proposto di scaricare un file compresso corrispondente al file-immagine di un CD-ROM, da usare per installare la distribuzione di Plan 9. In generale conviene prelevare questo file, anche se esisterebbe la possibilità di eseguire l'installazione attraverso la rete. Il file-immagine va espanso e quindi si procede in qualche modo all'incisione del CD-ROM.

```
$ bunzip2 < plan9.iso.bz2 > plan9.iso [Invio]
```

Il comando mostrato viene eseguito idealmente in un sistema Unix; il file '`plan9.iso.bz2`' è il file compresso originale e '`plan9.iso`' è il file-immagine da usare per l'incisione del CD-ROM.

Quando si dispone del CD-ROM e del dischetto di avvio, si è pronti per iniziare l'installazione, ma bisogna mettere in conto la possibilità di qualche piccolo imprevisto. Per esempio, c'è da considerare

il fatto che Plan 9 è progettato per essere usato quasi esclusivamente attraverso un'interfaccia grafica e così funziona anche la procedura di installazione. Ma è proprio la grafica che crea i maggiori problemi di compatibilità ed è una delle caratteristiche importanti da stabilire quando ci si accinge a prelevare il file-immagine del dischetto di avvio (e del CD-ROM). Anche se le caratteristiche della grafica sono state scelte con cura, può darsi che il dischetto di avvio mostri una grafica annebbiata o invisibile per qualunque ragione; in questo modo, diventa impossibile procedere all'installazione. Se si incontrano problemi del genere, si può provare a intervenire in un file di configurazione contenuto nel dischetto, ovviamente con l'aiuto di un altro sistema operativo funzionante.

Il dischetto che si ottiene contiene un file system Dos-VFAT; comune e il file da modificare è '`plan9.ini`':

```
*nomp=1
distname=plan9
nobootprompt=local!/boot/bzroot

# serial console on COM1
#console=0

# disable ether and usb probing
#*nousbprobe=1
#*noetherprobe=1

monitor=lcd
vgasize=800x600x8
mouseport=ps2
bootfile=fd0!dos!9pcflop.gz
installurl=http://204.178.31.2/magic/9down4e/compressed/↩
↪1061467246.r7pet673tqud33uuhxt6hed5yd7k7ae9
```

Si osservi che il file in questione utilizza un codice di interruzione di riga pari a *<CR><LF>* e tale deve rimanere, anche se viene modificato attraverso un sistema Unix.

Nel file si può notare il riferimento al «numero di serie» (che in realtà è una stringa) necessario per procedere all'installazione attraverso la rete, incorporato nello stesso indirizzo HTTP che verrebbe usato per questo.

Per quanto riguarda il problema della grafica, si osservino le direttive seguenti:

```
monitor=lcd
vgasize=800x600x8
```

Come si può intendere, il dischetto è stato richiesto per uno schermo LCD (di un elaboratore portatile), con una risoluzione bassa (800×600). Se l'immagine che si ottiene non è ben visibile, si possono mettere qui valori comuni, facendo qualche tentativo, per esempio indicando un monitor '**`vga`**' puro e semplice; inoltre, alle volte può succedere che anche una profondità di colori troppo bassa sia la causa di una grafica offuscata e inutilizzabile:

```
monitor=vga
vgasize=800x600x16
```

Alla fine, se si comprende che si può anche aumentare la risoluzione, tanto meglio:

```
monitor=vga
vgasize=1024x768x16
```

La configurazione corretta del dischetto è cruciale, perché è quella che poi viene trasferita durante installazione nel disco fisso.

In situazioni di difficoltà estreme, si può mettere un valore impossibile nella direttiva '**`vgasize`**', in modo tale che la procedura di installazione non riesca ad avviare la grafica. In questo modo si ha la possibilità di avviare il programma di installazione senza la grafica, ma questo poi si riflette anche nel sistema che viene installato e purtroppo Plan 9 è difficile da manovrare senza la grafica (almeno inizialmente).

Eventualmente, il file '`plan9.ini`' può contenere anche un menù, secondo una modalità che ricorda quella del file '`CONFIG.SYS`' del Dos. Per il momento viene mostrato un esempio senza spiegazioni, che dovrebbe risultare abbastanza comprensibile a livello intuitivo. Una configurazione del genere (adattando eventualmente ciò che riguarda le interfacce di rete), dovrebbe servire per evitare di perdere troppo tempo nei tentativi che si fanno, per esempio quando magari si è convinti che il mouse sia collegato su una porta seriale e invece si trova sull'altra:

```
*nomp=1
distname=plan9
nobootprompt=local!/boot/bzroot

[menu]
menuitem=1,  multisync75 1024x768x8  mouse 0    ether rtl8139
menuitem=2,  multisync75 1024x768x8  mouse 1    ether rtl8139
menuitem=3,  multisync75 1024x768x8  mouse ps2  ether rtl8139
menuitem=4,  multisync75 1024x768x8  mouse 0    ether ne2000 port 300 irq 11
menuitem=5,  multisync75 1024x768x8  mouse 1    ether ne2000 port 300 irq 11
menuitem=6,  multisync75 1024x768x8  mouse ps2  ether ne2000 port 300 irq 11
menuitem=7,  vga         800x600x8   mouse 0    ether rtl8139
menuitem=8,  vga         800x600x8   mouse 1    ether rtl8139
menuitem=9,  vga         800x600x8   mouse ps2  ether rtl8139
menuitem=10, vga         800x600x16  mouse 0    ether rtl8139
menuitem=11, vga         800x600x16  mouse 1    ether rtl8139
menuitem=12, vga         800x600x16  mouse ps2  ether rtl8139

[1]
ether0=type=rtl8139
monitor=multisync75
vgasize=1024x768x8
mouseport=0

[2]
ether0=type=rtl8139
monitor=multisync75
vgasize=1024x768x8
mouseport=1

[3]
ether0=type=rtl8139
monitor=multisync75
vgasize=1024x768x8
mouseport=ps2

[4]
ether0=type=ne2000 port=0x300 irq=11
monitor=multisync75
vgasize=1024x768x8
mouseport=0

[5]
ether0=type=ne2000 port=0x300 irq=11
monitor=multisync75
vgasize=1024x768x8
mouseport=1

[6]
ether0=type=ne2000 port=0x300 irq=11
monitor=multisync75
vgasize=1024x768x8
mouseport=ps2

[7]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x8
mouseport=0

[8]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x8
mouseport=1

[9]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x8
mouseport=ps2

[10]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x16
mouseport=0

[11]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x16
mouseport=1

[12]
ether0=type=rtl8139
monitor=vga
vgasize=800x600x16
mouseport=ps2

[common]
bootfile=fd0!dos!9pcflop.gz
installurl=http://204.178.31.2/magic/9down4e/compressed/↩
↪1061467246.r7pet673tqud33uuhxt6hed5yd7k7ae9
```

Installazione nel disco fisso

«

Quando si dispone finalmente di un dischetto che funziona correttamente con il tipo di grafica di cui si dispone, assieme al CD-ROM si può procedere all'installazione nel disco fisso. L'installazione richiede in pratica la creazione di una partizione Plan 9, all'interno della quale la procedura definisce automaticamente un gruppo di sottopartizioni, dove poi viene copiata la distribuzione contenuta nel CD-ROM. Si richiede la disponibilità di almeno 2 Gibyte liberi.

Si osservi che il dischetto non deve essere protetto contro la scrittura, perché vengono creati dei file al suo interno.

Viene mostrato un esempio di questa installazione, su un elaboratore contenente un disco fisso con una partizione Dos-FAT comune e dello spazio libero da usare per creare la partizione Plan 9.

Una volta avviato l'elaboratore attraverso il dischetto, dopo alcuni messaggi diagnostici, se tutto va bene si avvia la grafica, all'interno della quale appare una finestra di terminale, da dove si esegue l'installazione. Inizialmente dovrebbe apparire la presentazione seguente:

```
Preparing menu...
The following unfinished tasks are ready to be done:
    configfs    - choose the type of file system to install
    stop        - save the current installation state, to be resumed later

Task to do [configfs]:
```

In pratica si tratta di un menù molto semplice, dal quale si intende anche la possibilità di salvare nel dischetto lo stato della procedura di installazione, nel caso si decida di sospenderla.

L'invito del menù suggerisce l'operazione più appropriata (appare tra parentesi quadre) e se corrisponde a ciò che si vuole scegliere, è sufficiente premere [*Invio*] senza scriverla. Qui comunque si mostra il procedimento completo:

```
Task to do [configfs]: configfs[Invio]

You can install the following types of file systems:

    fossil      an archival (dump) file server
    kfs         the old Plan 9 on-disk file server

If you choose to install fossil, you can add Venti later,
assuming you leave enough disk space for it.

file system (fossil, kfs)[fossil]:
```

Viene richiesto così di specificare il tipo di file system da utilizzare; l'invito ha questa volta una forma differente, perché tra parentesi tonde sono elencate le opzioni disponibili. Inizialmente può essere più conveniente l'utilizzo del tipo di file system Kfs, che è il più vecchio ma più documentato:

```
file system (fossil, kfs)[fossil]: kfs[Invio]

Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install

The following unfinished tasks are ready to be done:
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    stop        - save the current installation state, to be resumed later

Task to do [partdisk]:
```

Come si vede, la fase successiva richiede di intervenire nella definizione delle partizioni. Il programma che si ottiene è veramente spartano:

```
Task to do [partdisk]: partdisk[Invio]

The following disk devices were found.

sdC0 - IBM-DBCA-204860
 * p1                   0 261       (261 cylinders, 1.99 GB) FATHUGE
   empty              261 582       (331 cylinders, 2.53 GB)

cdC1 - TEAC CD-224E 1.5A

Disk to partition (sdC0, sdC1)[no default]:
```

Poco prima di arrivare a questo punto è stato inserito il CD-ROM nel lettore che corrisponde alla seconda unità ATA; pertanto, vengono

indicati due dischi, ma quello che conta è il primo, '**sdC0**', in cui appare già una partizione Dos-FAT:

```
Disk to partition (sdC0, sdC1)[no default]: sdC0[Invio]

This is disk/fdisk; use it to create a Plan 9 partition.
If there is enough room, a Plan 9 partition will be
suggested; you can probably just type 'w' and then 'q'.

cylinder = 8225280 bytes
 * p1                   0 261       (261 cylinders, 1.99 GB) FATHUGE
'  p2                 261 592       (331 cylinders, 2.53 GB) PLAN9
>>>
```

Leggendo la descrizione si comprende che lo spazio libero è già stato utilizzato per una nuova partizione di tipo Plan 9 e sarebbe sufficiente salvare le cose così per proseguire. A ogni modo, conviene dare un'occhiata ai comandi disponibili all'interno del programma '`disk/fdisk`', per scoprire che si tratta di un programma molto simile, operativamente, a '**`fdisk`**' di un sistema GNU/Linux:

```
>>> h[Invio]

. [newdot] - display or set value of dot
a name [start [end]] - add partition
d name - delete partition
h - print help message
p - print partition table
w - write partition table
q - quit
A name - set partition active
t name [type] - set partition type

>>>
```

Come accennato, in questo caso basta salvare le modifiche e uscire dal programma:

```
>>> w[Invio]

>>> q[Invio]

Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)

The following unfinished tasks are ready to be done:
    prepdisk    - subdivide plan 9 disk partition
    stop        - save the current installation state, to be resumed later

Task to do [prepdisk]:
```

Come si può vedere, viene proposto di passare alla suddivisione della partizione in sottopartizioni, per gli scopi di Plan 9:

```
Task to do [prepdisk]: prepdisk[Invio]

The following Plan 9 disk partitions were found.

/dev/sdC0/plan9
  empty                0 5317515    (5317515 sectors, 2.53 GB)

Plan 9 partition to subdivide (/dev/sdC0/plan9)[/dev/sdC0/plan9]:
```

Come si vede, viene proposto di suddividere la partizione, che appare indicata attraverso una forma simile a quella dei file di dispositivo dei sistemi Unix:

```
Plan 9 partition to subdivide ↩
↪(/dev/sdC0/plan9)[/dev/sdC0/plan9]: /dev/sdC0/plan9[Invio]
```

Ovviamente, se non si sta usando una tastiera con la disposizione dei tasti secondo la mappa USA, la barra obliqua va cercata per tentativi.

```
This is disk/prep; use it to subdivide the Plan 9 partition.
If it is not yet subdivided, a sensible layout will be suggested;
you can probably just type 'w' and then 'q'.

no plan9 partition table found
9fat 204800
nvram 1
fs 4090171
swap 1022543
' 9fat                0 204800     (204800 sectors, 100.00 MB)
' nvram          204800 204801     (1 sectors, 512 B)
' fs              204801 4294972    (4090171 sectors, 1.95 GB)
' swap           4294972 5317515    (1022543 sectors, 499.28 MB)
>>>
```

Come si può comprendere, il programma '`disk/prep`' appena avviato si comporta in modo simile a '`disk/fdisk`':

```
>>> w[Invio]

>>> q[Invio]
```

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition

The following unfinished tasks are ready to be done:
    mountfs     - choose and mount file system partition
    stop        - save the current installation state, to be resumed later
```

```
Task to do [mountfs]: mountfs[Invio]
```

A questo punto si innesta il file system appena creato nella sottopartizione '`/dev/sdC0/fs`':

```
The following partitions named fs* were found.

Please choose one to use as the installation file system
for your Plan 9 installation.

--rw-r---- S 0 glenda glenda 2094167552 Aug 11 15:29 /dev/sdC0/fs
```

```
Kfs partition ↩
↪(/dev/sdC0/fs)[/dev/sdC0/fs]: /dev/sdC0/fs[Invio]
```

```
A Plan 9 kfs file system already exists on /dev/sdC0/fs.
Do you want to wipe it clean?
```

```
Wipe the Plan 9 file system clean (keep, wipe)[keep]: wipe
[Invio]
```

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition

The following unfinished tasks are ready to be done:
    configdist  - choose the source of the distribution archive
    download    - download or continue to download the distribution archive
    stop        - save the current installation state, to be resumed later

Task to do [configdist]:
```

A questo punto si deve selezionare in che modo prelevare la distribuzione. Dal momento che è stato predisposto un CD-ROM, si sceglie la prima ipotesi. Si ricorda che in questo esempio il CD-ROM si trova inserito in un lettore che corrisponde alla seconda unità ATA:

```
Task to do [configdist]: configdist[Invio]
```

```
Are you going to download the distribution
from the internet or do you have it on local media?
```

```
Distribution is from (local, net)[local]: local[Invio]
```

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition
    configdist  - choose the source of the distribution archive

The following unfinished tasks are ready to be done:
    mountdist   - locate and mount the distribution
    download    - download or continue to download the distribution archive
    stop        - save the current installation state, to be resumed later
```

```
Task to do [mountdist]: mountdist[Invio]
```

```
Please wait... Scanning storage devices...
    /dev/sdC0/9fat
    /dev/sdC0/data
    /dev/sdC0/dos
    /dev/sdC0/fs
    /dev/sdC0/nvram
    /dev/sdC0/swap
    /dev/sdC1/data

The following storage media were detected.
Choose the one containing the distribution.

    /dev/sdC1/data (iso9660 cdrom)
```

```
Distribution disk [no default]: /dev/sdC1/data[Invio]
```

```
Which directory contains the distribution?
Any of the following will suffice (in order of preference):
    - the root directory of the cd image
    - the directory containing plan9.iso
    - the directory containing plan9.iso.bz2
Typing 'browse' will put you in a shell that you can use to
look for the directory.

Location of archive [browse]:
```


La prima volta, questa richiesta potrebbe non apparire chiara: si tratta di indicare dove si trova la copia del contenuto del CD-ROM. Se il CD-ROM è stato ottenuto espandendo il file-immagine relativo e poi incidendo, basta specificare che questo si articola a partire dalla radice, ovvero da '`/`'; diversamente si indica una directory particolare. Si osservi che il CD-ROM, o un disco qualunque che sia accessibile (anche una partizione con un file system Ext2 o Ext3 di un sistema GNU/Linux può andare bene), potrebbe contenere al suo interno soltanto il file-immagine, denominato '`plan9.iso`'; teoricamente andrebbe bene anche il file-immagine compresso, denominato '`plan9.iso.bz2`', ma in pratica il sistema di installazione potrebbe non riuscire a espanderlo. Anche così si può procedere all'installazione e la directory da indicare si intende essere quella che contiene questo file:

```
Location of archive [browse]: /[Invio]
```

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition
    configdist  - choose the source of the distribution archive
    mountdist   - locate and mount the distribution

The following unfinished tasks are ready to be done:
    copydist    - copy the distribution into the file system
    stop        - save the current installation state, to be resumed later

Task to do [copydist]:
```

Ecco che si può così passare alla copia della distribuzione; operazione che richiede un po' di tempo, ma una barra di attesa mostra il progredire dell'operazione.

```
Task to do [copydist]: copydist[Invio]
```

Passa un po' di tempo...

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition
    configdist  - choose the source of the distribution archive
    mountdist   - locate and mount the distribution
    copydist    - copy the distribution into the file system

The following unfinished tasks are ready to be done:
    bootsetup   - create a boot floppy or configure hard disk to boot plan 9
    stop        - save the current installation state, to be resumed later

Task to do [bootsetup]:
```

A questo punto è il momento di predisporre un sistema di avvio. Per sicurezza conviene realizzare anche un dischetto apposito:

```
Task to do [bootsetup]: bootsetup[Invio]
```

```
Initializing Plan 9 FAT configuration partition (9fat)

add 9load at clust 2
Initializing FAT file system
type hard, 12 tracks, 255 heads, 63 sectors/track, 512 bytes/sec
Adding file /n/kfs/386/9load. length 180364
add 9load at clust 2
used 184320 bytes
used 0 bytes


There are myriad ways to boot a Plan 9 system.
You can use any of the following.
(You can also repeat this task to use more than one).

  floppy - create a boot floppy
  plan9  - make the plan 9 disk partition the default for booting
  win9x  - add a plan 9 option to windows 9x boot menu
  winnt  - add a plan 9 option to windows nt/2000/xp boot manager

(See the documentation for instructions on booting Plan 9 from LILO.)

If you are upgrading an extant third edition installation and booting
from something other than a floppy, you needn't run anything here.
Just type ctl-d.
Enable boot method (floppy, plan9, win9x, winnt)[no default]:
```

Viene resa avviabile la partizione:

```
Enable boot method ↩
↪(floppy, plan9, win9x, winnt)[no default]: plan9[Invio]
```

```
Your Plan 9 partition is more than 2GB into your disk,
and the master boot record used by Windows 9x/ME
cannot access it (and thus cannot boot it).

You can install the Plan 9 master boot record, which can load
partitions far into the disk.

If you use the Windows NT/2000/XP master boot record
or a master boot record from a Unix clone (e.g. LILO or
FreeBSD bootmgr, it is probably safe to continue using
that boot record rather than install the Plan 9 boot
record.

Install the Plan 9 master boot record (y, n)[no default]:
```

In questo caso si preferisce installare il settore di avvio di Plan 9, dato che la partizione Dos-FAT preesistente non contiene nulla e non ci sono altri sistemi operativi da avviare:

```
Install the Plan 9 master boot record (y, n)[no default]: y
```
[*Invio*]

```
Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition
    configdist  - choose the source of the distribution archive
    mountdist   - locate and mount the distribution
    copydist    - copy the distribution into the file system
    bootsetup   - create a boot floppy or configure hard disk to boot plan 9

The following unfinished tasks are ready to be done:
    finish      - finish the installation and reboot
    stop        - save the current installation state, to be resumed later

Task to do [finish]:
```

Si preferisce creare un dischetto di avvio prima di concludere:

```
Task to do [finish]: bootsetup
```
[*Invio*]

```
There are myriad ways to boot a Plan 9 system.
You can use any of the following.
(You can also repeat this task to use more than one).

  floppy - create a boot floppy
  plan9  - make the plan 9 disk partition the default for booting
  win9x  - add a plan 9 option to windows 9x boot menu
  winnt  - add a plan 9 option to windows NT/2000/XP boot manager

(See the documentation for instructions on booting Plan 9 from LILO.)

If you are upgrading an extant third edition installation and booting
from something other than a floppy, you needn't run anything here.
Just type ctl-d.
```

```
Enable boot method ↩
↪(floppy, plan9, win9x, winnt)[no default]: floppy
```
[*Invio*]

```
Insert a disk other than your installation boot disk
into your floppy drive; it will be erased to create
the boot floppy.

Press enter when ready.
```

Viene così estratto il dischetto usato per l'installazione e se ne inserisce un altro, già inizializzato a basso livello.

```
Press enter when ready.
```
[*Invio*]

```
add 9load at clust 2
add 9pcdisk.gz at clust 163
add plain9ini.bak at clust 6c1
Initializing FAT file system
type 3½HD, 80 traks, 2 heads, 18 sectors/track, 512 bytes/sec
Adding file /n/kfs/386/9load. length 180364
add 9pcdisk.gz at clust 163
add plain9ini.bak at clust 6c1
used 885760 bytes

Done!


Preparing menu...
The following tasks are done:
    configfs    - choose the type of file system to install
    partdisk    - edit partition table (e.g., to create a plan 9 partition)
    prepdisk    - subdivide plan 9 disk partition
    mountfs     - choose and mount file system partition
    configdist  - choose the source of the distribution archive
    mountdist   - locate and mount the distribution
    copydist    - copy the distribution into the file system
    bootsetup   - create a boot floppy or configure hard disk to boot plan 9

The following unfinished tasks are ready to be done:
    finish      - finish the installation and reboot
    stop        - save the current installation state, to be resumed later
```

```
Task to do [finish]: finish
```
[*Invio*]

Al termine conviene rimettere il disco usato per l'installazione:

```
We need to write the state of the current installation to the install floppy,
so that you can pick up from here if, for example, you want to set up
more boot methods.
```

```
Please make sure the install floppy is in the floppy drive and
press enter.
```
[*Invio*]

```
Your install state has been saved to the install floppy.

Congratulations; you've completed the install.

Halting file systems...
done

Remember to take the install disk out of the drive.
Feel free to turn off your computer.
```

Per verificare si può riavviare con la combinazione di tasti [*Ctrl Alt Canc*].

Avvio e arresto del sistema installato

Una volta riavviato l'elaboratore, eventualmente con l'aiuto del dischetto di avvio preparato durante l'installazione, si arriva a un punto in cui appare una domanda che può sembrare incomprensibile:

```
root is from (il, tcp, local)[local!#S/sdC0/fs]:
```

Intuitivamente si comprende che si tratta della possibilità di avviare il sistema operativo dal disco locale o attraverso altri mezzi (per esempio la rete). È sufficiente confermare la voce suggerita per procedere all'avvio della copia locale.

```
root is from (il, tcp, local)[local!#S/sdC0/fs]:
```
[*Invio*]

Inizialmente, il sistema prevede l'utente '**glenda**', senza parola d'ordine:

```
user [none]: glenda
```
[*Invio*]

Si ottiene così l'avvio del sistema grafico, dove, in particolare, appare una finestra di terminale con in evidenza un documento che introduce all'uso di Plan 9.

Dal momento che si utilizza un file system Kfs, si arresta il sistema con il comando seguente:

```
% disk/kfscmd halt
```
[*Invio*]

```
kfs file system halted
```

Si osservi che se si sta utilizzando un file system diverso da Kfs (come nel caso di Fossil), si ottiene soltanto un messaggio di errore, del tipo: '**kfscmd: can't open command file**'.

Per poter impartire un comando del genere serve una finestra di terminale, che dovrebbe essere già disponibile la prima volta che si avvia il sistema installato.

In mancanza di altro, anche se il file system non è ancora stato fermato con un comando apposito, si può comunque riavviare il sistema con la combinazione di tasti [*Ctrl Alt Canc*].

Riferimenti

- Lucent Technologies, *Plan 9 from Bell Labs*
  http://plan9.bell-labs.com/plan9dist/

[1] **Plan 9** Lucent Public License

Caratteristiche dei primi sistemi UNIX e BSD 2100
Partizioni 2100
Dispositivi 2101
File di dispositivo delle unità di memorizzazione 2101
Unità a nastro 2102
Collocazione dei file eseguibili 2102
SIMH e il PDP-11 2102
Utilizzo normale 2102
Avvio e sospensione di una simulazione 2103
Nastri virtuali 2103
Caratteristiche generali dei vari modelli PDP-11 2107
Simulazione unità a disco e a nastro 2108
2.11BSD 2109
Preparazione del nastro virtuale 2109
Configurazione iniziale del simulatore e avvio 2110
Preparazione delle partizioni 2111
Inizializzazione del file system 2114
Copia dei file principali e primo avvio del sistema 2115
Sistemazione dell'avvio dal disco 2116
Predisposizione dei file di dispositivo e conclusione dell'installazione 2117
Sistemare la data 2118
Installazione di file-immagine pronti 2118
UNIX versione 5 (RK05) 2118
UNIX versione 6 (RK05) 2119
UNIX versione 6 (RL02) 2120
UNIX versione 7 (RL02) 2122
UNIX versione 7 (RL02) «Torsten» 2124
BSD versione 2.9 (RL02) 2126
Derivazioni di UNIX per hardware ridotto 2129
Mini-UNIX 2129
LSI UNIX o LSX 2130
Programmi di servizio 2131
V7fs 2131
Riferimenti 2132

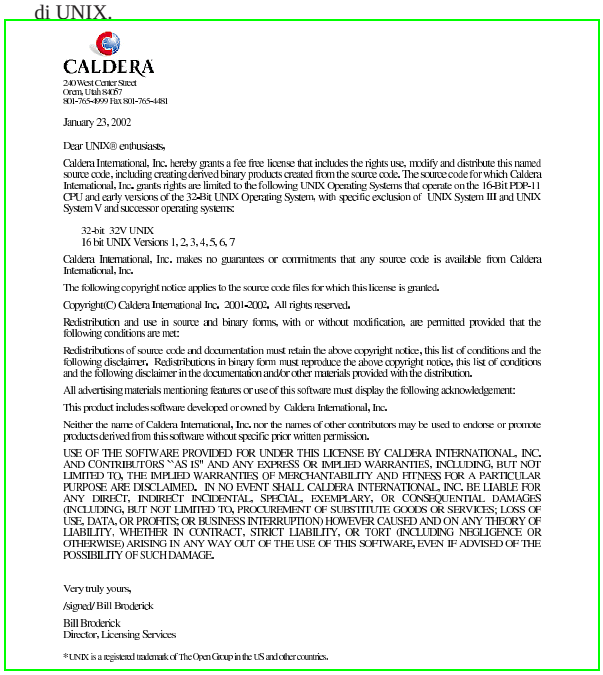
UNIX nasce negli elaboratori PDP-11. La diffusione e lo studio del sistema operativo ha favorito la fortuna di questa linea di elaboratori, tanto che esiste ancora un discreto interesse verso la preservazione dell'hardware del PDP-11 e delle architetture derivate.

Negli anni 1970 UNIX è un sistema operativo di ricerca, dal quale emerge la filosofia della condivisione della conoscenza informatica. Il desiderio di preservare il PDP-11 va di pari passo con quello di conservare, per ciò che è possibile, quanto resta di tali versioni di Unix, assieme alle varianti ed estensioni dell'università di Berkeley, dello stesso periodo.

La proprietà dei diritti sul codice UNIX originale si è trasferita più volte. Ogni «proprietario» ha gestito --o preteso di gestire-- questi diritti a propria discrezione; in particolare si ricorda nel 1994 la controversia tra Novell e l'università di Berkeley e nel 2003 quella tra SCO Group e IBM. Ma nel 2002 va ricordato un evento importante: Caldera (successivamente divenuta SCO Group), la quale in quel momento ne aveva i diritti, rilascia le prime edizioni di UNIX con una licenza simile a quella di BSD. Purtroppo, nel momento in cui SCO Group ha iniziato la causa contro IBM, questa licenza è scomparsa dal sito originale, ma continua a essere conservata e pubblicata dagli amatori del vecchio UNIX. Si veda a questo proposito:

- Caldera, *Dear UNIX® enthusiast*, 2002
  http://minnie.tuhs.org/Archive/Caldera-license.pdf
- Dion L. Johnson II, *Liberal license for ancient UNIX sources*, 2002
  http://www.lemis.com/grog/UNIX/
  http://www.lemis.com/grog/UNIX/ancient-source-all.pdf

Figura u193.1. La licenza, in stile «BSD», per le prime versioni di UNIX.



CALDERA
240 West Center Street
Orem, Utah 84057
801-765-4999 Fax 801-765-4481

January 23, 2002

Dear UNIX® enthusiasts,

Caldera International, Inc. hereby grants a fee free license that includes the rights use, modify and distribute this named source code, including creating derived binary products created from the source code. The source code for which Caldera International, Inc. grants rights are limited to the following UNIX Operating Systems that operate on the 16-Bit PDP-11 CPU and early versions of the 32-Bit UNIX Operating System, with specific exclusion of UNIX System III and UNIX System V and successor operating systems:

32-bit 32V UNIX
16 bit UNIX Versions 1, 2, 3, 4, 5, 6, 7

Caldera International, Inc. makes no guarantees or commitments that any source code is available from Caldera International, Inc.

The following copyright notice applies to the source code files for which this license is granted.

Copyright(C) Caldera International Inc. 2001-2002. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code and documentation must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed or owned by Caldera International, Inc.

Neither the name of Caldera International, Inc. nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

USE OF THE SOFTWARE PROVIDED FOR UNDER THIS LICENSE BY CALDERA INTERNATIONAL, INC. AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL CALDERA INTERNATIONAL, INC. BE LIABLE FOR ANY DIRECT, INDIRECT INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Very truly yours,

/signed/ Bill Broderick

Bill Broderick
Director, Licensing Services

*UNIX is a registered trademark of The Open Group in the US and other countries.

Caratteristiche dei primi sistemi UNIX e BSD

Le prime versioni di UNIX e di BSD (quelle per il PDP-11) hanno delle caratteristiche comuni, anche se non si può avere la certezza che siano sempre perfettamente uniformi. Queste caratteristiche vengono riassunte in questo capitolo.

Partizioni

Un disco utilizzato da una distribuzione BSD tradizionale è sempre diviso in partizioni, mentre con UNIX queste non ci sono, in quanto lo spazio finale, dopo la conclusione del file system viene usato per lo scambio della memoria virtuale.

Le partizioni BSD sono identificate da «etichette», definite *disklabel*. I nomi di queste etichette sono dati da una lettera alfabetica seguita da due punti: «**x**:». Una distribuzione BSD tradizionale può gestire per ogni disco un massimo di otto partizioni e le prime hanno generalmente un ruolo prestabilito.

Tabella u193.2. Utilizzo normale delle partizioni nei sistemi BSD tradizionali.

Etichetta	Utilizzo
`a:`	Partizione principale.
`b:`	Partizione di scambio per la memoria virtuale.
`c:`	Spazio complessivo di tutto il disco.
`d:` `e:` `f:` `g:` `h:`	Partizioni disponibili.

Sempre per quanto riguarda BSD, la partizione identificata dall'etichetta '**`c:`**' serve generalmente a delimitare lo spazio complessivo del disco. Si osservi che non è possibile collocare una partizione per lo scambio della memoria virtuale in una partizione diversa dalla seconda; inoltre, tra i file di dispositivo non ne è previsto uno che rappresenti il disco complessivo: da ciò deriva la necessità di avere l'etichetta '**`c:`**'.

Dispositivi

Le distribuzioni UNIX e BSD tradizionali classificano i dispositivi di memorizzazione, che nel PDP-11 potevano essere molto diversi, in gruppi omogenei, in base all'organizzazione del codice necessario per accedervi. Questi gruppi hanno delle sigle che è necessario conoscere.

Tabella u193.3. Sigle e file di dispositivo usati per le unità di memorizzazione. Le associazioni sono indicative.

unità di controllo	modello	tipo	BSD	
RK11	RK05	cartridge	rk	/dev/**rk**0a /dev/r**rk**0a
RK611	RK06 RK07		hk	/dev/**hk**0a /dev/r**hk**0a
RL11 RLV12	RL01 RL02		rl	/dev/**rl**0a /dev/r**rl**0a
RP11	RP01 RP02 RP03	pack	rp	/dev/**xp**0a /dev/r**xp**0a
Massbus: RH70 RH11	RP04 RP05 RP06	pack	xp	/dev/**xp**0a /dev/r**xp**0a
	RP07	fixed		
	RM02	pack		
	RM03 RM05			
	RM80	fixed		
MSCP: RQDX3	RA60	pack	ra	/dev/**ra**0a /dev/r**ra**0a
	RA70 RA80 RA81 RA82 RA90 RA92	fixed		
	RD51 RD52 RD53 RD54 RD31 RD32	fixed		
	RX33	fd 5,25 in		
	RX50	fd 5,25 in		
RX11 RX211	RX01 RX02	fd 8 in		

unità di controllo	modello	tipo	BSD	
MSCP: RQDX3	RA60	pack	ra	/dev/**ra**0a /dev/r**ra**0a
	RA70 RA71 RA72 RA73 RA80 RA81 RA82 RA90 RA92	fixed		
	RD51 RD52 RD53 RD54 RD31 RD32	fixed		
	RX33	fd 5,25 in		
	RX50	fd 5,25 in		
RX11 RX211	RX01 RX02	fd 8 in		

TM02 TM03	TE16 TU45 TU77	tape	ht	
	TE10 TM11		tm	
	TS11 TSV05		ts	

File di dispositivo delle unità di memorizzazione

I file di dispositivo delle unità di memorizzazione a disco sono classificate in base al gruppo di dispositivi di cui fanno parte; inoltre, possono essere disponibili a coppie: una versione a blocchi e un'altra a caratteri.

Nello UNIX di ricerca, la struttura del nome dei file di dispositivo per le unità a disco segue una regola abbastanza semplice:

`[r]`***`hhn`***

La lettera «r» iniziale, se appare indica un dispositivo a caratteri e sta per *raw*. I dispositivi di questo tipo (a caratteri), se previsti, si usano solo nella fase di inizializzazione e creazione dei file system, oltre che nella copia brutale dell'immagine della partizione.

Le due lettere successive, che nel modello appaiono come ***hh***, richiamano il nome del tipo di dispositivo, come annotato nella tabella della sezione precedente. Per esempio, il file '`/dev/rk0`' è un dispositivo a blocchi per l'accesso al primo disco di tipo '**`rk`**' (un disco RK05). Nello stesso modo, il file '`/dev/rrk0`' è il dispositivo a caratteri (*raw*) dello stesso disco.

Nel caso di BSD, la forma del nome di questi file di dispositivo rimane la stessa, con l'aggiunta della lettera della partizione. Per esempio, il file '`/dev/ra0a`' è un dispositivo a blocchi per l'accesso diretto alla prima partizione del primo disco di tipo '**`ra`**'. Nello

stesso modo, il file '`/dev/rra0a`' è un dispositivo a caratteri della stessa partizione. È da osservare che con BSD un file di dispositivo a blocchi per il disco intero non è più disponibile, mentre rimane per la versione a caratteri; per esempio, esiste '`/dev/rhk0`', ma **non esiste** più '`/dev/hk0`'.

Unità a nastro

I file di dispositivo delle unità a nastro non sono differenziati e vanno ricreati al volo, in base al tipo di nastro effettivamente esistente.

Collocazione dei file eseguibili

Nei primi sistemi UNIX non esistono le directory '`/sbin/`' e '`/usr/sbin/`', in quanto i programmi più delicati si trovano invece nella directory '`/etc/`' che non è inclusa nei percorsi predefiniti per questo. Pertanto, per avviare programmi come '**`mknod`**' o '**`mkfs`**', occorre anteporre tutto il percorso:

```
# /etc/mknod …
```

SIMH e il PDP-11

SIMH[1] è un simulatore di hardware per una serie di vecchi elaboratori, tra i quali anche il famoso PDP-11. La simulazione implica generalmente l'accesso a file su disco che rappresentano, di volta in volta, l'immagine di un disco, di un nastro, di una stampante.

Per poter utilizzare SIMH occorre leggere la documentazione originale annotata alla fine del capitolo. Qui vengono annotate solo poche cose e in particolare ciò che riguarda il PDP-11.

Utilizzo normale

SIMH è costituito da un gruppo di programmi, ognuno specializzato per la simulazione di un certo tipo di elaboratore. Per esempio, per avviare la simulazione di un PDP-11, si usa normalmente il programma '**`pdp11`**'. Se si avvia il programma senza argomenti, si interagisce con il simulatore:

```
$ pdp11 [Invio]

  PDP-11 simulator V3.6-1

sim>
```

Da questa modalità interattiva, si danno dei comandi, con i quali si specificano delle opzioni di funzionamento e si definisce l'uso di file-immagine di unità che devono essere gestite. Di norma si prepara uno script per non perdere tempo, come nell'esempio seguente:

```
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
ATTACH  RL0  bsd_2.9_root_rl02.dsk
SHOW    RL0
;
; Boot.
;
BOOT    RL0
```

Le istruzioni che si possono dare dipendono molto dal tipo particolare di simulazione prescelto e sono documentate separatamente, rispetto alla guida generale sull'uso di SIMH. È comunque utile osservare che questi comandi non fanno differenza nell'uso di lettere maiuscole o minuscole, a parte quando si fa riferimento a file-immagine, che vanno scritti come richiede il sistema operativo esterno.

Per eseguire uno script è sufficiente avviare il programma seguito dal nome dello stesso:

```
$ pdp11 avvio.ini [Invio]
```

In alternativa, durante il funzionamento interattivo, è possibile usare il comando '**`DO`**':

```
sim> do avvio.ini [Invio]
```


Durante il funzionamento interattivo del simulatore, è possibile concludere l'attività con il comando '**`QUIT`**':

```
sim> quit [Invio]
```

Oltre al fatto che i comandi di SIMH possono essere espressi indifferentemente con lettere maiuscole o minuscole, va osservato che questi comandi possono essere abbreviati; pertanto, spesso si vedono esempi di utilizzo di SIMH con comandi apparentemente differenti, per il solo fatto che sono stati abbreviati.

Avvio e sospensione di una simulazione

In condizioni normali, salvo configurazione differente, SIMH associa alla combinazione di tasti [*Ctrl e*] la sospensione della simulazione. In pratica, una volta avviata la simulazione, questa combinazione ne produce la sospensione.

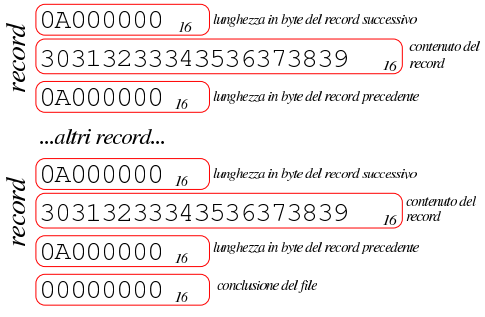
Una volta sospesa una simulazione, la si può riprendere, allo stato in cui si trovava, con il comando '**`CONT`**'; inoltre, è possibile salvare lo stato di funzionamento di una simulazione sospesa in un file, per poi recuperarla in un secondo momento e riprendere la simulazione da quella condizione.

Comando	Descrizione
[*Ctrl e*]	Sospende la simulazione.
`CONT`	Riprende la simulazione sospesa.
`SAVE` ***file***	Salva in un file una simulazione sospesa.
`RESTORE` ***file***	Recupera da un file una simulazione sospesa (da riprendere poi con il comando '**`CONT`**').

Nastri virtuali

SIMH gestisce i nastri magnetici come file su disco, aventi però una struttura particolare che riproduce l'organizzazione dei dati nel nastro stesso. Il nastro, per sua natura, è suddiviso in ***record***; pertanto, anche i file di SIMH devono riprodurre tale informazione.

Figura u193.7. Struttura dei dati che rappresentano un nastro virtuale per SIMH.



In pratica, ogni record (che deve avere una quantità pari di byte) è preceduto e anche seguito da 32 bit che rappresentano la dimensione dello stesso, in byte, tenendo conto che il valore deve essere rappresentato in modalità *little-endian*. Alla fine del file, altri 32 bit a zero indicano la conclusione dello stesso (come se fosse l'inizio di un record vuoto. L'esempio che appare nella figura mostra la sequenza di questi dati; in particolare, il numero $0A000000_{16}$, va interpretato effettivamente come $0000000A_{16}$ (a causa dell'inversione *little-endian*), pari a 10_{10}; pertanto, il record è composto da 10 byte.

Figura u193.8. Esempio di un nastro virtuale per SIMH, visto in esadecimale e secondo il codice ASCII. Il file originale è una sequenza di 100 byte contenenti le cifre numeriche da zero a nove (da 30_{16} a 39_{16} secondo il codice ASCII), suddiviso in record da 10 byte.

```
00000000  0a 00 00 00 30 31 32 33  34 35 36 37 38 39 0a 00  |....0123456789..|
00000010  00 00 0a 00 00 00 30 31  32 33 34 35 36 37 38 39  |......0123456789|
00000020  0a 00 00 00 0a 00 00 00  30 31 32 33 34 35 36 37  |........01234567|
00000030  38 39 0a 00 00 00 0a 00  00 00 30 31 32 33 34 35  |89........012345|
00000040  36 37 38 39 0a 00 00 00  0a 00 00 00 30 31 32 33  |6789........0123|
00000050  34 35 36 37 38 39 0a 00  00 00 0a 00 00 00 30 31  |456789........01|
00000060  32 33 34 35 36 37 38 39  0a 00 00 00 0a 00 00 00  |23456789........|
00000070  30 31 32 33 34 35 36 37  38 39 0a 00 00 00 0a 00  |0123456789......|
00000080  00 00 30 31 32 33 34 35  36 37 38 39 0a 00 00 00  |..0123456789....|
00000090  0a 00 00 00 30 31 32 33  34 35 36 37 38 39 0a 00  |....0123456789..|
000000a0  00 00 0a 00 00 00 30 31  32 33 34 35 36 37 38 39  |......0123456789|
000000b0  0a 00 00 00 00 00 00 00                           |........|
000000b8
```

SIMH offre solo strumenti di conversione da altri formati o di analisi del contenuto, ma manca la possibilità di creare un nastro a partire da file comuni e di estrarre poi i file stessi. Per questo occorre realizzare un proprio programma. Gli esempi che si vedono nei listati successivi, funzionano correttamente se la piattaforma prevede interi da 32 bit, rappresentati in modalità *little-endian*.

Listato u193.9. Programma per la conversione di un file comune nel formato adatto a SIMH per simulare i nastri magnetici.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;         // 32 bit, little endian.
    char acRecord[65535];       // 65535 is the max record length.
    int  iZero;
    int  iByteRead;
    int  iByteLeft;
    //
    iZero = '\0';
    //
    // Get the record length from command line argument.
    //
    sscanf (argv[1], "%d", &iRecordLength);
    //
    // Read and write data.
    //
    while (1)   // Loop forever.
      {
        //
        // Read from standard input one record.
        //
        iByteRead = fread (acRecord, 1, iRecordLength, stdin);
        //
        if (iByteRead == iRecordLength)
          {
            //
            // The record was read completely.
            //
            fwrite (&iRecordLength, 4, 1, stdout);
            fwrite (acRecord, iByteRead, 1, stdout);
            fwrite (&iRecordLength, 4, 1, stdout);
          }
        else if (iByteRead == 0)
          {
            //
            // Nothing was read. The file is finished.
            //
            fwrite (&iZero, 1, 4, stdout);
            break;
          }
        else if (iByteRead < iRecordLength)
          {
            //
            // The record was read partially: it must be
            // filled with zeroes.
            //
            iByteLeft = iRecordLength - iByteRead;
            //
            fwrite (&iRecordLength, 4, 1, stdout);
            fwrite (acRecord, iByteRead, 1, stdout);
            while (iByteLeft > 0)
              {
                fwrite (&iZero, 1, 1, stdout);
                iByteLeft--;
              }
            fwrite (&iRecordLength, 4, 1, stdout);
            //
            // The file is finished.
            //
            fwrite (&iZero, 1, 4, stdout);
            break;
          }
      }
    return 0;
}
```

Il programma che si vede nel listato precedente converte un file normale in un «file su nastro», leggendo lo standard input e generando il risultato attraverso lo standard output. Se il file si chiama '`convert_file_to_simh_tape.c`', si compila semplicemente così:

```
$ cc convert_file_to_simh_tape.c [ Invio ]
```

```
$ mv a.out convert_file_to_simh_tape [ Invio ]
```

Supponendo di volere convertire il file '`mio_file`' in un nastro virtuale, avente record da 1024 byte, si può procedere così:

```
$ ./convert_file_to_simh_tape 1024 < mio_file > mio_file.tap
[ Invio ]
```

Una volta convertiti tutti i file che si vogliono usare, si possono mettere assieme in uno stesso «nastro virtuale», semplicemente concatenandoli, per esempio così:

```
$ cat file_0.tap file_1.tap … > nastro_completo.tap [ Invio ]
```

L'estrazione di un file da un nastro richiede invece un procedimento più complesso. L'esempio riportato nel listato successivo mostra un programma che si limita a estrarre il primo file.

Listato u193.10. Programma per l'estrazione del primo file contenuto nell'immagine di un nastro virtuale di SIMH.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;         // 32 bit, little endian.
    char acRecord[65535];       // 65535 is the max record length.
    int  iZero;
    int  iByteRead;
    int  iByteLeft;
    //
    iZero = '\0';
    //
    // Read and write data.
    //
    while (1)  // Loop forever.
      {
        //
        // Read from standard input the record length.
        //
        iByteRead = fread (&iRecordLength, 1, 4, stdin);
        //
        if (iByteRead == 0)
          {
            //
            // The file is finished.
            //
            break;
          }
        else if (iByteRead < 4)
          {
            //
            // This should not happen.
            //
            return 1;
          }
        else if (iRecordLength == 0)
          {
            //
            // As the value is zero, this is the end of the
            // first file, and no other file is saved.
            //
            break;
          }
        else
          {
            //
            // Continue reading a record.
            //
            iByteRead = fread (acRecord, 1, iRecordLength, stdin);
            //
            if (iByteRead < iRecordLength)
              {
                //
                // The record is not complete.
                //
                return 1;
              }
            else
              {
                //
                // The record seems ok: write to output.
                //
                fwrite (acRecord, iByteRead, 1, stdout);
                //
                // Try to read from standard input the same
                // old record length (and ignore it).
                //
                iByteRead = fread (&iRecordLength, 1, 4, stdin);
                //
                if (iByteRead < 4)
                  {
```

```
                    //
                    // this should not happen!
                    //
                    return 1;
                  }
              }
          }
        //
        // Continue the loop.
        //
      }
    return 0;
}
```

Il programma appena mostrato, si aspetta di leggere un file-immagine conforme alle specifiche di SIMH e non è in grado di gestire altre situazioni. Se si verificano errori, il programma termina di funzionare senza avvertimenti di qualunque sorta. Il file in ingresso viene atteso dallo standard input e il file estratto viene emesso attraverso lo standard output. Se il file si chiama '`estrai.c`', si compila semplicemente così:

```
$ cc estrai.c [Invio]

$ mv a.out estrai [Invio]
```

Supponendo di volere estrarre il primo file contenuto nell'immagine '`mio_file.tap`' si può procedere così:

```
$ ./estrai < mio_file.tap > file_0 [Invio]
```

Dal momento che i dati in un nastro sono organizzati in record di dimensione uniforme, è normale che quanto estratto contenga qualche byte in più, ma a zero (00_{16}). Di norma, i file che vengono archiviati su nastro hanno una struttura tale per cui questa aggiunta diventa ininfluente.

Listato u193.11. Programma completo per l'estrazione di tutti i file da un'immagine di un nastro di SIMH.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    int  iRecordLength;         // 32 bit, little endian
    char acRecord[65535];       // 65535 is the max record length.
    int  iZero;
    int  iByteRead;
    int  iByteLeft;
    char aczRootFileName[252];
    char aczFileName[255];
    int  iFileCounter;
    FILE *pfOut;
    //
    iZero        = '\0';
    iFileCounter = '\0';
    //
    // Get root file name.
    //
    sscanf (argv[1], "%s", aczRootFileName);
    //
    // Open the first output file.
    //
    sprintf (aczFileName, "%s-%03d", aczRootFileName, iFileCounter);
    pfOut = fopen (aczFileName, "w");
    printf ("%s\n", aczFileName);
    //
    // Read and write data.
    //
    while (1)
      {
        //
        // Read from standard input the record length.
        //
        iByteRead = fread (&iRecordLength, 1, 4, stdin);
        //
        if (iByteRead == 0)
          {
            //
            // The file is finished, although it is not
            // correctly ended. There are no more files.
            //
            fclose (pfOut);
            //
            break;
          }
        else if (iByteRead < 4)
          {
            //
            // This should not happen, but close anyway.
            //
            fclose (pfOut);
            //
            return 1;
          }
        else if (iRecordLength == 0)
          {
```



```
            //
            // Then length of the next record is zero.
            // This is the end of the first file: prepare the next one.
            //
            fclose (pfOut);
            iFileCounter++;
            sprintf (aczFileName, "%s-%03d", aczRootFileName, iFileCounter);
            pfOut = fopen (aczFileName, "w");
            printf ("%s\n", aczFileName);
            //
          }
        else
          {
            //
            // The record length was read: no read the record.
            //
            iByteRead = fread (acRecord, 1, iRecordLength, stdin);
            //
            if (iByteRead < iRecordLength)
              {
                //
                // The record is not complete: close.
                //
                fwrite (acRecord, iByteRead, 1, pfOut);
                fclose (pfOut);
                //
                return 1;
              }
            else
              {
                //
                // The record seems ok: write to output.
                //
                fwrite (acRecord, iByteRead, 1, pfOut);
                //
                // Try to read from standard input the same
                // old record length (and ignore it).
                //
                iByteRead = fread (&iRecordLength, 1, 4, stdin);
                //
                if (iByteRead < 4)
                  {
                    //
                    // this should not happen: close.
                    //
                    fclose (pfOut);
                    //
                    return 1;
                  }
              }
          }
        //
        // Continue the loop.
        //
      }
    //
    return 0;
}
```

Il programma che appare nell'ultimo listato è completo, in quanto estrapola tutti i file contenuti in un'immagine di nastro secondo SIMH. Il programma riceve dalla riga di comando la radice del nome dei file da creare, quindi genera una sequenza numerata con quella radice. In generale, l'ultimo file è vuoto e va ignorato.

Se il programma del listato è contenuto nel file '`simh_tape_to_file.c`', si compila così:

```
$ cc convert_simh_tape_to_file.c [Invio]

$ mv a.out convert_simh_tape_to_file [Invio]
```

Supponendo di volere estrarre i file contenuti nell'immagine '`mio_file.tap`' si può procedere così:

```
$ ./convert_simh_tape_to_file radice < mio_file.tap [Invio]

  radice-000
  radice-001
  radice-002
  …
```

In tal caso si ottengono i file '`radice-000`', '`radice-001`' e così di seguito.

Caratteristiche generali dei vari modelli PDP-11

I modelli di PDP-11 che sono esistiti hanno avuto caratteristiche abbastanza varie. Anche se si intende utilizzare solo un simulatore, è necessario scegliere il modello adatto al sistema operativo che si vuole installare. Le cose più importanti da sapere sono i tipi di bus ammissibili e la dimensione massima della memoria centrale.

Tabella u193.13. Caratteristiche generali dell'unità centrale, in ordine di anno di produzione.

Modello	Bus	Memoria centrale	Anno
PDP11/15	Unibus	64 Kibyte	1970
PDP11/20	Unibus	64 Kibyte	1970
PDP11/10	Unibus	64 Kibyte	1972
PDP11/05	Unibus	64 Kibyte	1972
PDP11/45	Unibus, Fastbus	256 Kibyte	1972
PDP11/50	Unibus, Fastbus	256 Kibyte	1972
PDP11/55	Unibus, Fastbus	256 Kibyte	1972
PDP11/35	Unibus	256 Kibyte	1973
PDP11/40	Unibus	256 Kibyte	1973
PDP11/03	Q-bus	64 Kibyte	1975
PDP11/04	Unibus	64 Kibyte	1975
PDP11/70	Unibus, Massbus	4 Mibyte	1975
PDP11/34	Unibus	256 Kibyte	1976
PDP11/60	Unibus	256 Kibyte	1976
PDP11/23	Q-bus	4 Mibyte	1979
PDP11/44	Unibus, Private memory bus	4 Mibyte	1979
PDP11/24	Unibus	4 Mibyte	1980
PDP11/73	Q-bus	4 Mibyte	1983
PDP11/21	Q-bus	64 Kibyte	1983
PDP11/83	Q-bus, Private memory bus	4 Mibyte	1985
PDP11/84	Unibus, Private memory bus	4 Mibyte	1985
PDP11/53	Q-bus	4 Mibyte	1987
PDP11/93	Q-bus, Private memory bus	4 Mibyte	1990
PDP11/94	Unibus, Private memory bus	4 Mibyte	1990

Simulazione unità a disco e a nastro

SIMH offre la simulazione di una gamma di dischi e nastri magnetici abbastanza ampia. La tabella successiva associa i tipi principali di dischi e nastri simulati all'unità di controllo e alla sigla usata da SIMH.

Tabella u193.14. I tipi principali di dischi e nastri gestiti da SIMH, associati ai nomi usati dai primi sistemi UNIX e BSD. Lo spazio disponibile viene calcolato in modo compatibile all'uso consueto: byte e multipli di byte.

SIMH	unità di controllo	modello	tipo	BSD		capacità byte	byte/settore	settori/traccia	tracce/cilindro	cilindri
RK	RK11	RK05	cartridge	rk	/dev/rk0a /dev/rrk0a	2 494 464	512	12	2	203
HK	RK611	RK06		hk	/dev/hk0a /dev/rhk0a	13 888 512	512	22	3	411
		RK07				27 540 480	512	22	3	815
RL	RL11 RLV12	RL01		rl	/dev/rl0a /dev/rrl0a	5 242 880	256	40	2	256
		RL02				10 485 760	512	40	2	256
	RP11	RP01	pack	rp	/dev/rp0a /dev/rrp0a	5 196 800	512	5	10	203
		RP02				20 787 200	512	10	20	203
		RP03				40 960 000	512	10	20	400
RP	Massbus: RH70 RH11	RP04	pack	xp	/dev/xp0a /dev/rxp0a	79 964 160	512	20	19	411
		RP05				79 964 160	512	20	19	411
		RP06				174 423 040	512	22	19	815
		RP07	fixed			516 096 000	512	50	32	630
		RM02	pack			67 420 160	512	32	5	823
		RM03				67 420 160	512	32	5	823
		RM05				256 196 608	512	32	19	823
		RM80	fixed			128 221 184	512	32	14	559
RQ	MSCP: RQDX3	RA60	pack	ra	/dev/ra0a /dev/rra0a	204 890 112	512	42	4	2382
		RA70	fixed			280 084 992	512	33	11	1507
		RA71				700 062 720	512	51	14	1915
		RA72				1 000 089 600	512	51	20	1915
		RA73				2 007 390 880	512	70	21	2667
		RA80				121 325 568	512	31	14	546
		RA81				456 228 864	512	51	14	1248
		RA82				622 932 480	512	57	15	1423
		RA90				1 216 590 336	512	69	13	2649
		RA92				1 505 926 656	512	69	13	3279
		RD51	fixed			10 653 696	512	17	4	306
		RD52				30 965 760	512	18	7	480
		RD53				75 497 472	512	18	8	1024
		RD54				159 936 000	512	17	15	1225
		RD31				21 411 840	512	17	4	615
		RD32				42 823 680	512	17	6	820
		RX33	fd 5,25 in			1 228 800	512	15	2	80
						737 280	512	9	2	80
						409 600	512	10	1	80
						368 640	512	9	2	40
		RX50	fd 5,25 in			409 600	512	10	1	80
RX	RX11	RX01	fd 8 in			256 256	128	26	1	77
RY	RX211	RX02				512 512	256	26	1	77

TU	TM02 TM03	TE16	tape	ht	
		TU45			
		TU77			
TM		TE10		tm	
		TM11			
TS		TS11		ts	
		TSV05			

2.11BSD

Questo capitolo descrive un procedimento per installare 2.11BSD in un PDP-11 simulato attraverso SIMH, utilizzato a sua volta in un sistema GNU/Linux.

La lettura di questo capitolo è utile solo dopo quella della guida originale di Steven Schultz, *Installing and operating 2.11BSD on the PDP-11*, che può essere ottenuta dall'indirizzo *http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.pdf* .

I file usati per l'installazione descritta in questo capitolo provengono da *http://minnie.tuhs.org/Archive/PDP-11/Distributions/ucb/2.11BS11D/* . Eventualmente è disponibile anche una forma diversa della stessa distribuzione che parte direttamente da un file-immagine del disco contenente il file system principale, con tutto ciò che serve, già pronto per SIMH: *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.11_on_Simh/* .

Preparazione del nastro virtuale

Tra i file che compongono la distribuzione, quelli che servono per costruire il nastro di installazione sono i seguenti:

File	Descrizione
`'mtboot'`	Contiene il settore di avvio da usare con il nastro.
`'boot'`	Contiene il programma di avvio per la gestione iniziale del nastro.
`'disklabel'`	Contiene il programma, da avviare attraverso il nastro, usato per suddividere i dischi in partizioni.
`'mkfs'`	Contiene il programma, da avviare attraverso il nastro, con il quale inizializzare le partizioni dei dischi.
`'restore'`	Contiene il programma, da avviare attraverso il nastro, con il quale è possibile recuperare i dati archiviati con '**`dump`**'.
`'icheck'`	Contiene il programma, da avviare attraverso il nastro, per il controllo del file system di una partizione.
`'root.dump'`	Contiene l'archivio del file system principale, da estrarre con '**`restor`**'.
`'file6.tar.gz'`	Contiene l'archivio di quanto si articola a partire da '`/usr/`'.
`'file7.tar.gz'`	Contiene l'archivio di quanto si articola a partire da '`/usr/src/`'.
`'file8.tar.gz'`	Contiene l'archivio di altro materiale che si articola sempre a partire da '`/usr/src/`'.

I tre file con estensione '`.tar.gz`' vanno decompressi:

GNU/Linux `$ gunzip file*.tar.gz` [*Invio*]

I primi due file vanno combinati assieme, in modo da formare un solo file organizzato nel modo seguente:

`'mtboot'`	512 byte
`'mtboot'`	512 byte
`'boot'`	35 328 byte

In pratica, essendo i file più corti delle dimensioni indicate, vanno completati con byte a zero; inoltre, come si vede, il settore di avvio va duplicato. Si procede nel modo seguente per aggiustare la dimensione del primo file:

GNU/Linux `$ dd if=mkboot count=1 bs=512 conv=sync` ↩
↪ `of=mtboot.sync` [*Invio*]

Con il comando appena mostrato si ottiene il file '`mkboot.sync`' che completa la dimensione di 512 byte, come richiesto per il concatenamento. Il comando successivo predispone il file di avvio da collocare nella prima posizione del nastro:

GNU/Linux `$ cat mkboot.sync mkboot.sync boot > avvio` [*Invio*]

Si può quindi procedere alla preparazione del nastro virtuale, tenendo conto della dimensione dei settori prevista dalla documentazione originale, utilizzando i programmi descritti nella sezione u0.2:

```
GNU/Linux$ convert_file_to_simh_tape 512 < avvio > tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 1024 < disklabel ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 1024 < mkfs ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 1024 < restor ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 1024 < icheck ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 10240 < root.dump ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 10240 < file6.tar ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 10240 < file7.tar ↩
↪     >> tm11_0.tap [Invio]

GNU/Linux$ convert_file_to_simh_tape 10240 < file8.tar ↩
↪     >> tm11_0.tap [Invio][2]
```

Con gli strumenti di SIMH è possibile controllare il contenuto del nastro virtuale generato:

```
GNU/Linux$ mtdump tm11_0.tap [Invio]
```

Configurazione iniziale del simulatore e avvio

Si decide di simulare un PDP-11/44 del 1979, con solo 1 Mibyte di memoria centrale (il minimo per poter utilizzare 2.11BSD), con due unità a nastro connesse a un'unità di controllo TM11 (se si volesse usare l'unità TS11 si potrebbe gestire un solo nastro) e con un disco MSCP RA82 di dimensione inusuale: 200000000 byte. Un disco così capiente consente di installare tutto in una sola partizione, senza bisogno di innestarne altri.

Il file-immagine del disco deve essere creato prima di avviare il simulatore:

```
GNU/Linux$ dd if=/dev/zero of=ra82_0.dsk bs=1000000 count=200 [Invio]
```

Così facendo viene creato il file '`ra82_0.dsk`'. Nella simulazione vengono usati inoltre i file '`tm11_0.tap`', creato precedentemente con il necessario per procedere all'installazione del sistema, e '`tm11_1.tap`', il cui scopo è quello di disporre di un'unità ulteriore per archiviare dati mentre si usa 2.11BSD nel simulatore. Il file-immagine del secondo nastro non va predisposto, perché viene creato contestualmente al suo utilizzo.

L'ultima fase prima dell'avvio del simulatore consiste nel predisporre uno script con la configurazione desiderata della simulazione:

```
;
; PDP-11/44 (1979) with only 1 Mibyte RAM memory.
;
SET    CPU  11/44
SET    CPU  1024K
SHOW   CPU
;
; Devices that might be disabled.
;
;SET    RK   DISABLE
;SET    HK   DISABLE
;SET    TC   DISABLE
;SET    TS   DISABLE
;
; TM11 tape simulator.
;
SET    TM   ENABLED
SET    TM0  LOCKED
ATTACH TM0  tm11_0.tap
SHOW   TM0
;
SET    TM1  WRITEENABLED
ATTACH TM1  tm11_1.tap
SHOW   TM1
;
; MSCP disk.
;
; The actual disk has an unusual size: 200000000 byte
;
SET    RQ   ENABLED
SET    RQ0  RAUSER=200
ATTACH RQ0  ra82_0.dsk
SHOW   RQ0
;
; Should boot manually.
;
```

Supponendo che questo file si chiami '**`simh.ini`**', si può avviare il simulatore nel modo seguente:

```
GNU/Linux$ pdp11 simh.ini [Invio]

  PDP-11 simulator V3.5-1
  Disabling XQ
  CPU, 11/44, FPP, NOCIS, autoconfiguration on, 1024KB
  TM0, attached to tm11_0.tap, write locked, SIMH format
  TM: creating new file
  TM1, attached to tm11_1.tap, write enabled, SIMH format
  RQ0, 200MB, attached to ra82_0.dsk, write enabled, RAUSER
```

Lo script non contiene l'istruzione di avvio ('**`BOOT`**') che così deve essere data a mano. Ciò consente di verificare la correttezza della configurazione dai messaggi che si ottengono. Dall'invito del simulatore si può dare il comando di avvio attraverso il nastro contenente la distribuzione:

```
SIM> sim> BOOT TM0 [Invio]

  44Boot from tm(0,0,0) at 0172522

tape boot: :
```

Il programma di avvio contenuto nel primo file del nastro viene eseguito e si presenta così un altro invito (i due punti), dove va scritto quale programma eseguire (quale file eseguire) all'interno del nastro.

Preparazione delle partizioni

Dall'invito del programma di avvio della distribuzione occorre iniziare selezionando il programma che consente di predisporre le partizioni all'interno del disco virtuale. Questo programma ('**`disklabel`**') si trova nel secondo file del nastro (è il secondo in base alla procedura descritta per la preparazione di tale nastro); pertanto si avvia così:

```
tape boot: : tm(0,1) [Invio][3]

  Boot: bootdev=0401 bootscr=0172522
  disklabel

disklabel Disk?
```

A questo punto appare l'invito di '**`disklabel`**', dal quale è necessario inserire le coordinate del disco che si vuole suddividere in partizioni. In questo caso, si tratta di un'unità «ra», pertanto si usa la sigla '**`ra(0,0)`**':

```
disklabel Disk? ra(0,0) [Invio]

  'ra(0,0)' is unlabeled or the label is corrupt.

disklabel Proceed?  [y/n] y

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? y
```

Si comincia visualizzando la situazione, per poter calcolare la posizione delle partizioni:

```
disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? d
```

```
type: MSCP
disk: RA82
label: DEFAULT
flags:
bytes/sector: 512
sectors/track: 57
tracks/cylinder: 15
sectors/cylinder: 855
cylinders: 457
rpm: 3600
drivedata: 0 0 0 0 0

1 partitions:
#  size    offset  fstype   [fsize bsize]
a: 390800  0       2.11BSD   1024  1024    # (Cyl. 0 - 457*)
```

La partizione '**a:**' viene creata automaticamente, ma va modificata perché le dimensioni non sono corrette e perché occorre comunque una partizione per lo scambio della memoria virtuale.

In base alla geometria del disco, sono disponibili 457 cilindri contenenti 855 settori da 512 byte; pertanto, ogni cilindro ha una capacità di 437760 byte. 2.11BSD può utilizzare una sola partizione per lo scambio della memoria virtuale, al massimo da 32 Mibyte; pertanto, si possono utilizzare al massimo 76 cilindri per questo fine, pari a 64980 settori, mentre i restanti 381 cilindri, pari a 325755 settori, vanno usati per il file system:

```
disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? m

  modify

disklabel d(isplay) g(eometry) m(isc) p(artitions) q(uit)? p

  modify partitions

disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? a

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? t

disklabel 'a' fstype [2.11BSD]: 2.11BSD [Invio]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o

disklabel 'a' offset [0]: 0 [Invio][4]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s

disklabel 'a' size [390800]: 325755 [Invio][5]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? f

disklabel 'a' frags/fs-block [1]: 1 [Invio]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? F

disklabel 'a' frag size [1024]: 1024 [Invio]

  modify partition 'a'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? q

  modify partitions
```

Termina così la configurazione della partizione '**a:**'. Si può passare a '**b:**', che **deve** essere usata per lo scambio della memoria virtuale.


```
disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? b

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? t

disklabel 'b' fstype [unused]: swap [Invio][6]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o

disklabel 'b' offset [0]: 325755 [Invio][7]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s

disklabel 'b' size [0]: 64980 [Invio]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? f

disklabel 'b' frags/fs-block [1]: 1 [Invio]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? F

disklabel 'b' frag size [1024]: 1024 [Invio]

  modify partition 'b'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? q

  modify partitions
```

Termina anche la configurazione della partizione '**b:**' e si deve controllare che i dati inseriti siano coerenti, soprattutto che non ci siano accavallamenti tra le due partizioni.

```
disklabel d(isplay) n(umber) s(elect) q(uit)? d

  type: MSCP
  disk: RA82
  label: DEFAULT
  flags:
  bytes/sector: 512
  sectors/track: 57
  tracks/cylinder: 15
  sectors/cylinder: 855
  cylinders: 457
  rpm: 3600
  drivedata: 0 0 0 0 0

  2 partitions:
  #  size    offset  fstype   [fsize bsize]
  a: 325755  0       2.11BSD   1024 1024     # (Cyl. 0 - 380)
  b: 64980   325755  swap                    # (Cyl. 381 - 456)

  modify partitions
```

Si può procedere quindi con la partizione '**c:**', il cui scopo è solo quello di descrivere lo spazio usato complessivamente dalle altre due partizioni.

```
disklabel d(isplay) n(umber) s(elect) q(uit)? s

disklabel a b c d e f g h q(uit)? c

  sizes and offsets may be given as sectors, cylinders
  or cylinders plus sectors:  6200, 32c, 19c10s respectively
  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? o
```

```
disklabel 'c' offset [0]: 0[Invio]

  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? s

disklabel 'c' size [0]: 390735[Invio]

  modify partition 'c'

disklabel d(isplay) z(ero) t(ype) o(ffset) s(ize) f(rag) F(size)
q(uit)? q

  modify partitions
```

Si controlla ulteriormente la situazione:

```
disklabel d(isplay) n(umber) s(elect) q(uit)? d

  type: MSCP
  disk: RA82
  label: DEFAULT
  flags:
  bytes/sector: 512
  sectors/track: 57
  tracks/cylinder: 15
  sectors/cylinder: 855
  cylinders: 457
  rpm: 3600
  drivedata: 0 0 0 0 0

  2 partitions:
  #  size    offset  fstype   [fsize bsize]
  a: 325755  0       2.11BSD   1024 1024     # (Cyl. 0 - 380)
  b: 64980   325755  swap                    # (Cyl. 381 - 456)
  c: 390735  0       unused    1024 1024     # (Cyl. 0 - 456)

  modify partitions
```

A questo punto si può concludere confermando la suddivisione stabilita:

```
disklabel d(isplay) n(umber) s(elect) q(uit)? q

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? w

disklabel d(isplay) D(efault) m(odify) w(rite) q(uit)? q
```

Si ritorna così sotto il controllo del programma di gestione del nastro:

```
  44Boot from tm(0,0,1) at 0172522
tape boot:
```

Inizializzazione del file system

Il programma che serve a inizializzare il file system nella prima partizione del disco si trova nel terzo file del nastro. Il programma in question è '**mkfs**':

```
tape boot: tm(0,2)[Invio]

  Boot: bootdev=0402 bootcsr=0172522
  Mkfs

mkfs file system: ra(0,0)[Invio]
```

Si osservi che le coordinate '**ra(0,0)**' rappresentano precisamente la prima partizione del disco. Viene proposta la dimensione del file system, che è corretta, perché si riferisce a unità da 1024 byte (si perde un settore, perché la partizione ne è composta da una quantità dispari).

```
mkfs file sys size [162877]: [Invio]
```

Si conferma anche la dimensione degli inode:

```
mkfs bytes per inode [4096]: [Invio]
```

Per quanto riguarda la sequenza dei settori nel disco, trattandosi di una simulazione in un file, non serve a nulla che questi siano alternati, pertanto si evita tale accorgimento:

```
mkfs interleaving factor (m; 2 default): 1[Invio]
mkfs interleaving modulus (n; 427 default): 1[Invio]

  m/n = 1 1
  Exit called

  44Boot from tm(0,0,2) at 0172522
tape boot:
```

Terminata l'inizializzazione, si può fare la verifica del file system con il programma '**icheck**' che si trova nel quinto file del nastro:

```
tape boot: tm(0,4)[Invio]

  Boot: bootdev=0404 bootcsr=0172522
  Icheck

icheck File: ra(0,0)[Invio]

  ra(0,0):
  Not enough core; duplicates unchecked
  files 3 (r=1,d=2,b=0,c=0,l=0,s=0)
  used 2 (i=0,ii=0,iii=0,d=2)
  free 160328

  44Boot from tm(0,0,4) at 0172522
tape boot:
```

Copia dei file principali e primo avvio del sistema

Il nastro contiene quattro file separati da cui estrarre il contenuto del sistema operativo. Il primo, collocato nella sesta posizione del nastro, è un archivio ottenuto con il programma '**dump**' e contiene i file principali indispensabili per l'avvio di un sistema minimo; il secondo, collocato nella settima posizione, contiene ciò che va installato a partire dalla directory '`/usr/`'; il terzo, collocato nell'ottava posizione, contiene i sorgenti del kernel da installare a partire da '`/usr/src/`'; infine, il quarto, collocato nella nona posizione, contiene gli altri sorgenti disponibili e va installato sempre a partire da '`/usr/src/`'.

Si comincia con il ripristino dei file principali; poi, le operazioni successive si devono svolgere con il sistema avviato regolarmente.

```
tape boot: tm(0,3)[Invio]
```

Viene caricato il programma per il ripristino dei dati archiviati: '**restore**'.

```
  Boot: bootdev=0403 bootcsr=0172522
  Restor

restor Tape?: tm(0,5)[Invio]

restor Disk?: ra(0,0)[Invio]

restor Last chance before scribbling on disk. [Invio]

  End of tape

  44Boot from tm(0,0,3) at 0172522
tape boot:
```

A questo punto si può avviare il sistema minimo appena installato, per poi proseguire con le altre fasi di copia della distribuzione.

```
tape boot: ra(0,0)unix[Invio]
```

Quanto appena scritto indica di avviare il file '`unix`' che si trova nella directory radice del file system collocato nella prima partizione del primo disco. Il file '`unix`' è quindi il kernel del sistema.

```
  Boot: bootdev=02400 bootcsr=0172150

  2.11 BSD UNIX #115: Sat Apr 22 19:07:25 PDT 2000
  sms1@curly.2bsd.com:/usr/src/sys/GENERIC

  ra0: Ver 3 mod 6
  ra0: RA82  size=390800

  phys mem  = 1048576
  avail mem = 824640
  user mem  = 307200
```

```
June  8 21:21:24 init: configure system

hk 0 csr 177440 vector 210 attached
ht ? csr 172440 vector 224 skipped:  No CSR.
ra 0 csr 172150 vector 154 vectorset attached
rl 0 csr 174400 vector 160 attached
tm 0 csr 172520 vector 224 attached
tms 0 csr 174500 vector 260 vectorset attached
ts ? csr 172520 vector 224 interrupt vector already in use.
xp 0 csr 176700 vector 254 attached
erase, kill ^U, intr ^C
```

A questo punto appare l'invito del sistema operativo e si deve procedere con l'installazione degli altri archivi. È da osservare che il primo nastro magnetico viene individuato dal file di dispositivo '`/dev/rmt12`' (ed eventualmente il secondo corrisponde a '`/dev/rmt13`'), mentre le partizioni '***x*:**' corrispondono ai file di dispositivo '`/dev/rra0x`'. Ma occorre prima accertarsi che i file di dispositivo siano quelli adatti all'hardware scelto.

Sistemazione dell'avvio dal disco

Per il momento, il sistema è stato avviato con l'aiuto del programma di gestione del nastro. In un secondo momento, il nastro può essere usato ancora per avviare il disco, ma se è possibile, è meglio sistemare il programma di avvio all'inizio del disco:

2.11BSD# **cd /mdec** [*Invio*]

2.11BSD# **ls -l** [*Invio*]

```
total 14
-r--r--r--  1 root          512 Dec  5  1995 bruboot
-r--r--r--  1 root          512 Dec  5  1995 dvhpuboot
-r--r--r--  1 root          512 Dec  5  1995 hkuboot
-r--r--r--  1 root          512 Dec  5  1995 hpuboot
-r--r--r--  1 root          512 Dec  5  1995 rauboot
-r--r--r--  1 root          512 Dec  5  1995 rkuboot
-r--r--r--  1 root          512 Dec  5  1995 rluboot
-r--r--r--  1 root          512 Dec  5  1995 rm03uboot
-r--r--r--  1 root          512 Dec  5  1995 rm05uboot
-r--r--r--  1 root          512 Dec  5  1995 rx01uboot
-r--r--r--  1 root          512 Dec  5  1995 rx02uboot
-r--r--r--  1 root          512 Dec  5  1995 si51uboot
-r--r--r--  1 root          512 Dec  5  1995 si94uboot
-r--r--r--  1 root          512 Dec  5  1995 si95uboot
```

Dei programmi contenuti nella directory '`/mdec/`' occorre scegliere quello adatto al tipo di disco che si utilizza. In questo caso, si deve scegliere il file '`rauboot`'. Il file, della dimensione di un solo settore, va copiato con l'aiuto di '**dd**', all'inizio della prima partizione:

2.11BSD# **dd if=/mdec/rauboot of=/dev/rra0a count=1** [*Invio*]

```
1+0 records in
1+0 records out
```

A questo punto conviene verificare che l'operazione abbia avuto successo, arrestando il sistema (quello installato nell'hardware simulato con SIMH):

2.11BSD# **shutdown -h now** [*Invio*]

```
Shutdown at 18:02 (in 0 minutes) [pid 16]
#
System shutdown time has arrived
syncing disks... done
halting

HALT instruction, PC: 000014 (MOV #1,17406)
```

A questo punto si ritorna sotto il controllo di SIMH e si può tentare di avviare il sistema operativo direttamente dal disco:

SIMH sim> **BOOT RQ0** [*Invio*]

Si osservi che per SIMH, la sigla '**RQ0**' rappresenta il primo disco simulato, in base alla scelta dell'hardware fatta in precedenza.


```
44Boot from ra(0,0,0) at 0172150
```

disk boot: **ra(0,0)unix** [*Invio*]

```
Boot: bootdev=02400 bootcsr=0172150

2.11 BSD UNIX #115: Sat Apr 22 19:07:25 PDT 2000
sms1@curly.2bsd.com:/usr/src/sys/GENERIC

ra0: Ver 3 mod 6
ra0: RA82  size=390800

phys mem  = 1048576
avail mem = 824640
user mem  = 307200

hk 0 csr 177440 vector 210 attached
ht ? csr 172440 vector 224 skipped:  No CSR.
ra 0 csr 172150 vector 154 vectorset attached
rl 0 csr 174400 vector 160 attached
tm 0 csr 172520 vector 224 attached
tms 0 csr 174500 vector 260 vectorset attached
ts ? csr 172520 vector 224 interrupt vector already in use.
xp 0 csr 176700 vector 254 attached
erase, kill ^U, intr ^C
```

2.11BSD#

Predisposizione dei file di dispositivo e conclusione dell'installazione

Nella directory '`/dev/`' occorre eliminare i file di dispositivo riferiti alle unità a nastro, per ricrearli in base alle caratteristiche del tipo di nastro simulato effettivamente:

2.11BSD# **cd /dev** [*Invio*]

2.11BSD# **rm *mt*** [*Invio*]

2.11BSD# **./MAKEDEV tm0** [*Invio*][8]

2.11BSD# **sync** [*Invio*]

A questo punto, si può procedere con il recupero degli archivi rimasti.

2.11BSD# **cd /usr** [*Invio*]

Si riavvolge il nastro:

2.11BSD# **mt -f /dev/rmt12 rew** [*Invio*]

Si posiziona il nastro all'inizio della settima posizione:

2.11BSD# **mt -f /dev/rmt12 fsf 6** [*Invio*]

Si estrae l'archivio a partire dalla directory corrente:

2.11BSD# **tar xpbf 20 /dev/rmt12** [*Invio*]

Si procede in modo analogo per gli altri archivi.

2.11BSD# **mkdir /usr/src** [*Invio*]

2.11BSD# **cd /usr/src** [*Invio*]

In questo caso basta portare il nastro all'inizio del file successivo:

2.11BSD# **mt -f /dev/rmt12 fsf** [*Invio*]

2.11BSD# **tar xpbf 20 /dev/rmt12** [*Invio*]

Dopo aver estratto i sorgenti del kernel, occorre sistemare un collegamento simbolico:

2.11BSD# **cd /** [*Invio*]

2.11BSD# **rm -f /sys** [*Invio*]

2.11BSD# **ln -s usr/src/sys /sys** [*Invio*]

L'ultimo archivio da estrarre nella stessa directory '`/usr/src/`':

2.11BSD# **cd /usr/src** [*Invio*]

2.11BSD# **mt -f /dev/rmt12 fsf** [*Invio*]

2.11BSD# **tar xpbf 20 /dev/rmt12** [*Invio*]

Sistemare la data

Se si verifica la data, si può osservare che questa riporta l'anno 1995 e se si usa il comando '**date**', è possibile indicare solo le ultime due cifre dell'anno. Per risolvere il problema occorre un piccolo raggiro: prima si regola la data un secondo prima della mezzanotte del 1999, poi, passati al 2000, si può regolare l'ora in modo corretto:

```
2.11BSD # date [Invio]

  Fri Jun  9 11:52:39 PDT 1995

2.11BSD # date 9912312359.59 [Invio]

  Fri Dec 31 23:59:59 PST 1999

2.11BSD # date [Invio]

  Sat Jan  1 00:00:44 PST 2000

2.11BSD # date 0702041821 [Invio]

  Sun Feb  4 18:21:00 PST 2007
```

Si osservi che quando il sistema operativo (nell'hardware simulato) viene arrestato, l'orologio viene salvato e al riavvio successivo riprende da quel orario. Pertanto, a ogni riavvio occorre sistemare l'orologio.

```
2.11BSD # shutdown -h now [Invio]
```

Installazione di file-immagine pronti

Alcune edizioni dello UNIX di ricerca sono disponibili in file-immagine già pronti per questa o quella unità a disco. l'utilizzo di tali file con i simulatori è molto più semplice rispetto a una distribuzione su «nastro». Tuttavia, rimane il fatto che si tratta di versioni di UNIX prive di tanti accorgimenti a cui si è abituati se si conosce un sistema GNU e anche cose semplici come la correzione di quanto digitato sulla riga di comando possono essere impossibili.

UNIX versione 5 (RK05)

Si può trovare il file-immagine della versione 5 dello UNIX di ricerca, per un disco RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/research/Dennis_v5/v5root.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

```
GNU/Linux $ gunzip < v5root.gz > unix_v5_root_rk05.dsk [Invio]
```

Il nome scelto per il file estratto serve a sintetizzare le caratteristiche dell'immagine. Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RK05 cartridge disk.
;
SET     RK   ENABLE
ATTACH  RK0  unix_v5_root_rk05.dsk
SHOW    RK0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`unix_v5.ini`', si avvia la simulazione così:

```
GNU/Linux $ pdp11 unix_v5.ini [Invio]

  PDP-11 simulator V3.6-1
  Disabling XQ
  CPU, 11/45, FPP, autoconfiguration on, 256KB
  RK0, 1247KW, attached to unix_v5_root_rk05.dsk, write enabled
```

Quindi, dall'invito di SIMH si dà il comando di avvio:


```
SIMH sim> BOOT RK0 [Invio]
```

Se funziona, appare un altro invito, generato dal settore di avvio. Questo invito è rappresentato da una chiocciolina ('@'), dopo la quale va scritto il nome del file del kernel da eseguire:

```
disk boot @unix [Invio]
```

Praticamente non c'è alcuna procedura di avvio, quindi si ottiene immediatamente la richiesta di identificazione dell'utente:

```
UNIX login: root [Invio]

UNIX #
```

Si annotano i file di dispositivo presenti:

```
UNIX # chdir /dev [Invio]

UNIX # ls -l [Invio]

  total 0
  cr--r--r--  1 bin     1,  0 Nov 26 18:13 mem
  crw-rw-rw-  1 bin     1,  2 Nov 26 18:13 null
  crw--w--w-  1 root    0,  0 Mar 21 12:10 tty8
  #
```

Non è prevista la procedura di arresto del sistema ed è disponibile solo '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

```
UNIX # sync [Invio]

UNIX #  [Ctrl e]

  Simulation stopped, PC: 014150 (INC R4)

SIMH sim> quit [Invio]
```

UNIX versione 6 (RK05)

Si può trovare il file-immagine della versione 6 dello UNIX di ricerca, per un disco RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/research/Dennis_v6/v6root.gz* . Il file va estratto e quindi va preparato uno script per SIMH. Tuttavia, il settore di avvio contenuto nel file-immagine non funziona con il simulatore e va sostituito con una copia della versione 5.

Per prelevare il settore di avvio dal file-immagine della versione 5 si procede come nell'esempio seguente, dove si ottiene il file '`avvio`':

```
GNU/Linux $ dd if=unix_v5_root_rk05.dsk of=avvio bs=512 count=1
[Invio]
```

Si estrae il file che contiene l'immagine principale della versione 6:

```
GNU/Linux $ gunzip < v6root.gz > unix_v6_root_rk05_orig.dsk [Invio]
```

Si separa la porzione successiva al primo settore, generando un file temporaneo:

```
GNU/Linux $ dd if=unix_v6_root_rk05_orig.dsk of=tmp bs=512 skip=1
[Invio]
```

Si produce un nuovo file-immagine:

```
GNU/Linux $ cat avvio tmp > unix_v6_root_rk05_fixed.dsk [Invio]
```

Si prepara anche il secondo file-immagine previsto, contenente i sorgenti dei programmi:

```
GNU/Linux $ gunzip < v6src.gz > unix_v6_root_rk05_src.dsk [Invio]
```

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RK05 cartridge disks.
;
SET     RK   ENABLE
;
ATTACH  RK0  unix_v6_root_rk05_fixed.dsk
SHOW    RK0
```

```
;
ATTACH  RK1  unix_v6_src_rk05.dsk
SHOW    RK1
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v6.ini`’, si avvia la simulazione così:

GNU/Linux `$ pdp11 unix_v6.ini` [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RK0, 1247KW, attached to unix_v6_root_rk05_fixed.dsk, write enabled
RK1, 1247KW, attached to unix_v6_src_rk05.dsk, write enabled
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim> BOOT RK0` [*Invio*]

Se funziona appare l’invito del settore di avvio (‘`@`’), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta del kernel ‘`rkunix`’.

disk boot `@rkunix` [*Invio*]

UNIX `login: root` [*Invio*]

UNIX `#`

Si annotano i file di dispositivo presenti:

UNIX `# chdir /dev` [*Invio*]

UNIX `# ls -l` [*Invio*]

```
total 0
crw-rw-r--  1 bin     8,  1 May 13 20:01 kmem
crw-rw-r--  1 bin     8,  0 May 13 20:01 mem
crw-rw-rw-  1 bin     8,  2 May 13 20:01 null
crw--w--w-  1 root    0,  0 Aug 14 22:06 tty8
```

Non è prevista la procedura di arresto del sistema ed è disponibile solo ‘**sync**’, dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX `# sync` [*Invio*]

UNIX `#` [*Ctrl e*]

```
Simulation stopped, PC: 015670 (BNE 15722)
```

SIMH `sim> quit` [*Invio*]

UNIX versione 6 (RL02)

Si può trovare il file-immagine della versione 6 dello UNIX di ricerca, modificato per utilizzare un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/v6_rl02_unknown.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux `$ gunzip < v6_rl02_unknown.gz > unix_v6_root_rl02.dsk` [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v6_root_rl02.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file ‘`unix_v6.ini`’, si avvia la simulazione così:

GNU/Linux `$ pdp11 unix_v6.ini` [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v6_root_rl02.dsk, write enabled, RL02
```

Quindi, dall’invito di SIMH si dà il comando di avvio:

SIMH `sim> BOOT RL0` [*Invio*]

Se funziona appare l’invito del programma di avvio (‘`!`’), dal quale va scritto il nome del file del kernel da eseguire: ‘`unix`’.

disk boot `!unix` [*Invio*]

```
unix v6 11/23
mem = 99 KW max = 63
```

UNIX `#`

Da questo punto, il terminale potrebbe funzionare solo con lettere maiuscole, perciò conviene dare il comando successivo, in modo da ottenere le lettere minuscole consuete:

UNIX `# STTY -LCASE` [*Invio*]

Si annotano qui i file di dispositivo presenti, ma è comunque disponibile, nella directory ‘`/dev/`’ un file ‘`Makefile`’:

UNIX `# cd /dev` [*Invio*]

UNIX `# ls -l` [*Invio*]

```
crw-rw-r--  1 root      23,   0 Jun 19  1984 ad0
crw-rw-rw-  1 root      23,   1 Jul 25  1984 ad1
crw-rw-rw-  1 root      23,   2 Jul 25  1984 ad2
crw-rw-rw-  1 root      23,   3 Jul 25  1984 ad3
crw-rw-rw-  1 root      23,   4 Jul 25  1984 ad4
crw-rw-rw-  1 root      23,   5 Jul 25  1984 ad5
crw-rw-rw-  1 root      23,   6 Jul 25  1984 ad6
crw-rw-rw-  1 root      23,   7 Jul 25  1984 ad7
brw-rw-r--  1 root       6,   0 Aug  4  1982 hm0
brw-rw-r--  1 root       6,   1 Aug  4  1982 hm1
brw-rw-r--  1 root       6,   2 Aug  4  1982 hm2
brw-rw-r--  1 root       6,   3 Aug  4  1982 hm3
brw-rw-r--  1 root       6,   4 Aug  4  1982 hm4
brw-rw-r--  1 root       6,   5 Aug  4  1982 hm5
brw-rw-r--  1 root       6,   6 Aug  4  1982 hm6
brw-rw-r--  1 root       6,   7 Aug  4  1982 hm7
brw-rw-r--  1 root       6,   8 Aug  4  1982 hm8
crw-rw-r--  1 root       8,   3 Aug  4  1982 imem
crw-rw-r--  1 root       8,   1 Aug  4  1982 kmem
c-w--w--w-  1 root       2,   0 Aug  4  1982 lp
crw-rw-r--  1 root       8,   0 Aug  4  1982 mem
brw-rw-r--  1 root       3,   0 Aug  4  1982 mt0
crw-rw-r--  1 root       8,   2 Dec 16  1983 null
brw-rw-r--  1 root      15,   0 Aug  4  1982 rhm0
brw-rw-r--  1 root      15,   1 Aug  4  1982 rhm1
brw-rw-r--  1 root      15,   2 Aug  4  1982 rhm2
brw-rw-r--  1 root      15,   3 Aug  4  1982 rhm3
brw-rw-r--  1 root      15,   4 Aug  4  1982 rhm4
brw-rw-r--  1 root      15,   5 Aug  4  1982 rhm5
brw-rw-r--  1 root      15,   6 Aug  4  1982 rhm6
brw-rw-r--  1 root      15,   7 Aug  4  1982 rhm7
brw-rw-r--  1 root      15,   8 Aug  4  1982 rhm8
brw-rw-r--  1 root       0,   0 Aug  4  1982 rk0
brw-rw-r--  1 root       0,   1 Aug  4  1982 rk1
brw-rw-r--  2 root       2,   0 May  2  1983 rl0
brw-rw-r--  1 root       2,   1 May 12  1983 rl1
c-w--w--w-  1 root       2,   1 Aug  4  1982 rlp
brw-rw-r--  1 root       1,   0 Aug  4  1982 rp0
brw-rw-r--  1 root       1,   1 Aug  4  1982 rp1
brw-rw-r--  1 root       1,   2 Aug  4  1982 rp2
brw-rw-r--  1 root       1,   3 Aug  4  1982 rp3
crw-rw-r--  1 root      10,   0 Aug  4  1982 rrk0
crw-rw-r--  1 root      10,   1 Aug  4  1982 rrk1
crw-rw-r--  1 root      18,   0 May 26  1986 rrl0
crw-rw-r--  1 root      18,   1 Sep 12 12:35 rrl1
crw-rw-r--  1 root      12,   0 Aug  4  1982 rrp0
crw-rw-r--  1 root      12,   1 Aug  4  1982 rrp1
crw-rw-r--  1 root      12,   2 Aug  4  1982 rrp2
crw-rw-r--  1 root      12,   3 Aug  4  1982 rrp3
```

```
crw-rw-rw-  1 root      11,   0 Feb 25  1983 rrx0
crw-rw-rw-  1 root      11,   1 Feb 25  1983 rrx1
crw-rw-rw-  1 root      11,   2 Apr 26  1983 rrx2
crw-rw-rw-  1 root      11,   3 Feb 25  1983 rrx3
brw-rw-rw-  1 root       5,   0 Feb 25  1983 rx0
brw-rw-rw-  1 root       5,   1 Mar  2  1983 rx1
brw-rw-rw-  1 root       5,   2 Feb 25  1983 rx2
brw-rw-rw-  1 root       5,   3 Apr 26  1983 rx3
crw-rw-r--  1 root      21,   0 Aug  4  1982 stat0
crw-rw-r--  1 root      21,   1 Aug  4  1982 stat1
brw-rw-r--  2 root       2,   0 May  2  1983 swap
crw-rw-rw-  1 root       9,   0 Feb 14  1984 tty
crw--w--w-  1 root       0,   1 Feb  2 15:16 tty1
crw--w--w-  1 root       0,   2 Feb  2 09:51 tty2
crw--w--w-  1 root       0,   3 Feb  2 15:16 tty3
crw--w--w-  1 root       0,   4 Feb  2 15:16 tty4
crw--w--w-  1 root       0,   5 Feb  2 15:16 tty5
crw--w--w-  1 root       0,   0 Feb  2 15:25 tty8
crw-rw-rw-  1 root      20,   0 May  7  1983 ttya
crw-rw-rw-  1 root      20,   1 May  7  1983 ttyb
crw-rw-rw-  1 root      20,   2 May  7  1983 ttyc
crw-rw-rw-  1 root      20,   3 May  7  1983 ttyd
crw-rw-rw-  1 root      20,   4 May  7  1983 ttye
crw-rw-rw-  1 root      20,   5 May  7  1983 ttyf
crw-rw-rw-  1 root      20,   6 May  7  1983 ttyg
crw-rw-rw-  1 root      20,   7 May  7  1983 ttyh
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 017124 (CMPB #3,(R2))
```

SIMH sim> **quit** [*Invio*]

UNIX versione 7 (RL02)

Si può trovare il file-immagine della versione 7 dello UNIX di ricerca, per un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/v7_rl02_1145.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < v7_rl02_1145.gz > unix_v7_root_rl02.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v7_root_rl02.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`unix_v7.ini`', si avvia la simulazione così:

GNU/Linux $ **pdp11 unix_v7.ini** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v7_root_rl02.dsk, write enabled, RL02
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH sim> **BOOT RL0** [*Invio*]

Se funziona appare l'invito del settore di avvio ('@'), dal quale va scritto il nome del programma di avvio: '`boot`'.

disk boot **@boot** [*Invio*]

Quindi si inseriscono le coordinate del file del kernel da avviare:

```
New Boot, known devices are hp ht rk rl rp tm vt
```

disk boot : **rl(0,0)rl2unix** [*Invio*]

```
mem = 177856
```

UNIX #

Si annota il contenuto del file '`/dev/makefile`', con il quale si possono creare i file di dispositivo mancanti:

UNIX # **cd /dev** [*Invio*]

UNIX # **cat makefile** [*Invio*]

```
# You will want to do at least a make std (the default), followed b
# the make on the types of disks you have
#

std:
/etc/mknod console  c 0 0
/etc/mknod tty     c 17 0
/etc/mknod tty1     c 0 1
/etc/mknod tty2     c 0 2
/etc/mknod mem      c 8 0
/etc/mknod kmem     c 8 1
/etc/mknod null     c 8 2

rk:
/etc/mknod rk0  b 0 0
/etc/mknod rk1  b 0 1
/etc/mknod rrk0 c 9 0
/etc/mknod rrk1 c 9 1
chmod go-rw rk0 rk1 rrk0 rrk1

rl:
/etc/mknod rl0  b 8 0
/etc/mknod rl1  b 8 1
/etc/mknod rrl0 c 18 0
/etc/mknod rrl1 c 18 1
chmod go-rw rl0 rl1 rrl0 rrl1

rp03:
/etc/mknod rp0 b 1 1
/etc/mknod swap b 1 2
/etc/mknod rp3 b 1 3
/etc/mknod rrp0 c 11 1
/etc/mknod rrp3 c 11 3
chmod go-rw rp0 swap rp3 rrp0 rrp3

rp04 rp05:
/etc/mknod rp0 b 6 0
/etc/mknod swap b 6 1
/etc/mknod rp3 b 6 6
/etc/mknod rrp0 c 14 0
/etc/mknod rrp3 c 14 6
chmod go-rw rp0 swap rp3 rrp0 rrp3

rp06:
/etc/mknod rp0 b 6 0
/etc/mknod swap b 6 1
/etc/mknod rp3 b 6 7
/etc/mknod rrp0 c 14 0
/etc/mknod rrp3 c 14 7
chmod go-rw rp0 swap rp3 rrp0 rrp3

tm:
/etc/mknod mt0 b 3 0
/etc/mknod rmt0 c 12 0
/etc/mknod nrmt0 c 12 128
chmod go+w mt0 rmt0 nrmt0

ht:
/etc/mknod mt0 b 7 64
/etc/mknod mt1 b 7 0
/etc/mknod rmt0 c 15 64
/etc/mknod rmt1 c 15 0
/etc/mknod nrmt0 c 15 192
/etc/mknod nrmt1 c 15 128
chmod go+w mt0 mt1 rmt0 rmt1 nrmt0 nrmt1
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX `# sync` [*Invio*]

UNIX `#` [*Ctrl e*]

```
Simulation stopped, PC: 002312 (RTS PC)
```

SIMH `sim> quit` [*Invio*]

UNIX versione 7 (RL02) «Torsten»

Si può trovare il file-immagine della versione 7 dello UNIX di ricerca, per un disco RL02, modificato da Torsten Hippe, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/other/Torsten_Hippe_v7/v7.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux `$ gunzip < v7.gz > unix_v7_root_rl02_torsten.dsk` [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  unix_v7_root_rl02_torsten.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`unix_v7.ini`', si avvia la simulazione così:

GNU/Linux `$ pdp11 unix_v7.ini` [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to unix_v7_root_rl02_torsten.dsk, write enabled, RL02
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH `sim> BOOT RL0` [*Invio*]

Se funziona appare l'invito del settore di avvio ('@'), dal quale va scritto il nome del programma di avvio: '`boot`'.

disk boot `@boot` [*Invio*]

Quindi si inseriscono le coordinate del file del kernel da avviare:

```
Boot
```

disk boot `: rl(0,0)rl1unix` [*Invio*]

```
mem = 205376
```

Viene chiesto di eseguire un accesso normale. La parola d'ordine per l'utente '**root**' è «pdp». Inizialmente il terminale mostra solo lettere maiuscole:

UNIX `SINGLE USER LOGIN: ROOT` [*Invio*]

UNIX `PASSWORD: PDP` [*Invio*]

Con il comando successivo si riporta il terminale a funzionare con le lettere minuscole:

UNIX `# STTY -LCASE` [*Invio*]

Si annota il contenuto del file '`/dev/makefile`', con il quale si possono creare i file di dispositivo mancanti:

UNIX `# cd /dev` [*Invio*]

UNIX `# cat makefile` [*Invio*]


```
basic:
/etc/mknod console c 0 0
/etc/mknod tty c 1 0
/etc/mknod mem c 2 0
/etc/mknod kmem c 2 1
/etc/mknod null c 2 2
chmod go-w+r console
chmod go-w mem kmem
chmod go+rw null tty
chown bin mem kmem null tty
chgrp bin mem kmem null tty

rp:
make TYPE=rp bigdisk

hp:
make TYPE=hp bigdisk

rm:
make TYPE=rm bigdisk

hk:
make TYPE=hk bigdisk

si:
make TYPE=si bigdisk

bigdisk:
/etc/mknod $(TYPE)0 b 2 0
#       /etc/mknod $(TYPE)01 b 2 1
#       /etc/mknod $(TYPE)02 b 2 2
#       /etc/mknod $(TYPE)03 b 2 3
#       /etc/mknod $(TYPE)04 b 2 4
#       /etc/mknod $(TYPE)05 b 2 5
#       /etc/mknod $(TYPE)06 b 2 6
#       /etc/mknod $(TYPE)07 b 2 7
/etc/mknod $(TYPE)1 b 2 8
#       /etc/mknod $(TYPE)11 b 2 9
#       /etc/mknod $(TYPE)12 b 2 10
#       /etc/mknod $(TYPE)13 b 2 11
#       /etc/mknod $(TYPE)14 b 2 12
#       /etc/mknod $(TYPE)15 b 2 13
#       /etc/mknod $(TYPE)16 b 2 14
#       /etc/mknod $(TYPE)17 b 2 15
/etc/mknod r$(TYPE)0 c 11 0
#       /etc/mknod r$(TYPE)01 c 11 1
#       /etc/mknod r$(TYPE)02 c 11 2
#       /etc/mknod r$(TYPE)03 c 11 3
#       /etc/mknod r$(TYPE)04 c 11 4
#       /etc/mknod r$(TYPE)05 c 11 5
#       /etc/mknod r$(TYPE)06 c 11 6
#       /etc/mknod r$(TYPE)07 c 11 7
/etc/mknod r$(TYPE)1 c 11 8
#       /etc/mknod r$(TYPE)11 c 11 9
#       /etc/mknod r$(TYPE)12 c 11 10
#       /etc/mknod r$(TYPE)13 c 11 11
#       /etc/mknod r$(TYPE)14 c 11 12
#       /etc/mknod r$(TYPE)15 c 11 13
#       /etc/mknod r$(TYPE)16 c 11 14
#       /etc/mknod r$(TYPE)17 c 11 15
chmod go-w $(TYPE)[0-7] $(TYPE)[0-7]? r$(TYPE)[0-7] r$(TYPE)[0-7]?

rl:
/etc/mknod rl0 b 3 0
/etc/mknod rl1 b 3 1
/etc/mknod rrl0 c 12 0
/etc/mknod rrl1 c 12 1
chmod go-w rl? rrl?

rk:
/etc/mknod rk0 b 4 0
/etc/mknod rk1 b 4 1
/etc/mknod rrk0 c 13 0
/etc/mknod rrk1 c 13 1
chmod go-w rk? rrk?

dl:
/etc/mknod tty1 c 0 1
/etc/mknod tty2 c 0 2
chmod go-r+w tty[1-2]
< more 64% >dl:
/etc/mknod tty1 c 0 1
```

```
/etc/mknod tty2 c 0 2
chmod go-r+w tty[1-2]

pc:
/etc/mknod pc c 4 0
chown bin pc
chgrp bin pc
chmod go+w pc

lp:
/etc/mknod lp c 5 0
chown bin lp
chgrp bin lp
chmod go-rw lp

dc:
/etc/mknod dc0 c 6 0
/etc/mknod dc1 c 6 1
chown bin dc?
chgrp bin dc?
chmod go-rw dc?

dz:
/etc/mknod ttya c 7 0
/etc/mknod ttyb c 7 1
/etc/mknod ttyc c 7 2
/etc/mknod ttyd c 7 3
/etc/mknod ttye c 7 4
/etc/mknod ttyf c 7 5
/etc/mknod ttyg c 7 6
/etc/mknod ttyh c 7 7
chmod go-r+w tty[a-h]

tm:
/etc/mknod mt0 b 0 0
/etc/mknod mt1 b 0 1
/etc/mknod hmt0 b 0 64
/etc/mknod hmt1 b 0 65
/etc/mknod nmt0 b 0 128
/etc/mknod nmt1 b 0 129
/etc/mknod nhmt0 b 0 192
/etc/mknod nhmt1 b 0 193
/etc/mknod rmt0 c 10 0
/etc/mknod rmt1 c 10 1
/etc/mknod hrmt0 c 10 64
/etc/mknod hrmt1 c 10 65
/etc/mknod nrmt0 c 10 128
/etc/mknod nrmt1 c 10 129
/etc/mknod nhrmt0 c 10 192
/etc/mknod nhrmt1 c 10 193
chmod go-w *mt[01]

tc:
/etc/mknod tap0 b 1 0
/etc/mknod tap1 b 1 1
chmod go-w tap[01]

swap:
@echo ln ??? swap
```

Non essendo prevista la procedura di arresto del sistema, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 002312 (RTS PC)
```

SIMH sim> **quit** [*Invio*]

BSD versione 2.9 (RL02)

Si può trovare il file-immagine della versione 2.9 di BSD, per un disco RL02, presso *http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.9BSD_rl02_1145.gz* . Il file va estratto e quindi va preparato uno script per SIMH.

GNU/Linux $ **gunzip < 2.9BSD_rl02_1145.gz > bsd_2.9_root_rl02.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente:

```
;
; PDP-11/45 (1972) with only 256 Kibyte RAM memory.
;
SET     CPU  11/45
SHOW    CPU
;
; RL02 cartridge disks.
;
SET     RL   ENABLE
;
ATTACH  RL0  bsd_2.9_root_rl02.dsk
SHOW    RL0
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`bsd_6.9.ini`', si avvia la simulazione così:

GNU/Linux $ **pdp11 bsd_6.9.ini** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling XQ
CPU, 11/45, FPP, autoconfiguration on, 256KB
RL0, 5242KW, attached to bsd_2.9_root_rl02.dsk, write enabled, RL02
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH sim> **BOOT RL0** [*Invio*]

```
:boot

45Boot
```

Se funziona appare l'invito del programma di avvio ('**:**'), dal quale vanno scritte le coordinate per raggiungere il kernel da avviare:

disk boot : **rl(0,0)rlunix** [*Invio*]

```
Berkeley UNIX (Rev. 2.9.1) Sun Nov 20 14:55:50 PST 1983
mem = 135872

CONFIGURE SYSTEM:
xp 0 csr 176700 vector 254 attached
rk 0 csr 177400 vector 220 attached
hk 0 csr 177440 vector 210 attached
rl 0 csr 174400 vector 160 attached
rp ? csr 176700 vector 254 interrupt vector already in use
ht 0 csr 172440 vector 224 skipped:  No CSR
tm 0 csr 172520 vector 224 attached
ts 0 csr 172520 vector 224 interrupt vector already in use
dh ? csr 160020 vector 370 skipped:  No CSR
dm ? csr 170500 vector 360 skipped:  No autoconfig routines
dz ? csr 160110 vector 320 interrupt vector wrong
dz ? csr 160110 vector 320 interrupt vector wrong
dn 0 csr 175200 vector 300 skipped:  No autoconfig routines
vp ? csr 177500 vector 174 skipped:  No autoconfig routines
lp 0 csr 177514 vector 200 attached
Erase=^?, kill=^U, intr=^C
```

Si accede immediatamente alla shell. Si annota il contenuto della directory '`/dev/`'; tuttavia, è presente anche lo script '**MAKE**', con il quale si possono creare i file di dispositivo mancanti:

UNIX # **cd /dev** [*Invio*]

UNIX # **ls -l** [*Invio*]

```
total 5
-rwxrwxr-x 1 root     daemon      4375 Sep 19 01:21 MAKE
crw--w--w- 1 root     superuse   0,  0 Dec 31 16:03 console
brw------- 1 root     superuse   4,  0 Mar 29 15:42 hk0a
brw------- 1 root     superuse   4,  1 Mar 29 15:42 hk0b
brw------- 1 root     superuse   4,  2 Mar 29 15:42 hk0c
brw------- 1 root     superuse   4,  3 Mar 29 15:42 hk0d
brw------- 1 root     superuse   4,  6 Mar 29 15:42 hk0g
brw------- 1 root     superuse   4,  7 Mar 29 15:42 hk0h
crw------- 1 root     superuse   8,  1 Dec 31 16:02 kmem
crw------- 1 root     superuse   8,  0 Mar 29 15:40 mem
crw-rw-rw- 1 root     superuse   8,  2 Jul 27 17:58 null
crw------- 1 root     superuse  19,  0 Mar 29 15:42 rhk0a
crw------- 1 root     superuse  19,  1 Mar 29 15:42 rhk0b
crw------- 1 root     superuse  19,  2 Mar 29 15:42 rhk0c
crw------- 1 root     superuse  19,  3 Mar 29 15:42 rhk0d
```

```
crw------- 1 root     superuse  19,  6 Mar 29 15:42 rhk0g
crw------- 1 root     superuse  19,  7 Mar 29 15:42 rhk0h
brw------- 1 root     superuse   0,  0 Mar 29 15:40 rk0
brw------- 1 root     superuse   0,  1 Mar 30 00:14 rk1
brw------- 1 root     superuse   8,  0 Mar 29 15:40 rl0
brw------- 1 root     superuse   8,  1 Mar 30 00:14 rl1
brw-rw---- 1 root     daemon     6,  0 Mar 10 12:25 rm0a
brw-rw---- 1 root     daemon     6,  1 Mar 29 08:21 rm0b
brw------- 1 root     superuse   6,  2 Jun  2 09:41 rm0c
brw-rw---- 1 root     daemon     6,  3 Sep 29 18:13 rm0d
brw-rw---- 1 root     daemon     6,  4 Feb 25 04:58 rm0e
brw------- 1 root     superuse   1,  0 Mar 29 15:42 rp0a
brw------- 1 root     superuse   1,  1 Mar 29 15:43 rp0b
brw------- 1 root     superuse   1,  2 Mar 29 15:43 rp0c
brw------- 1 root     superuse   1,  7 Mar 29 15:43 rp0h
crw------- 1 root     superuse   9,  0 Jul 27 17:59 rrk0
crw------- 1 root     superuse   9,  1 Mar 30 00:14 rrk1
crw------- 1 root     superuse  18,  0 Dec 31 16:02 rrl0
crw------- 1 root     superuse  18,  1 Mar 30 00:14 rrl1
crw-rw---- 1 root     daemon    14,  0 Mar 15 18:30 rrm0a
crw-rw---- 1 root     daemon    14,  1 Jul 26 17:42 rrm0b
crw-rw-r-- 1 root     daemon    14,  2 Jan 11 21:19 rrm0c
crw-rw---- 1 root     daemon    14,  3 Mar 26 01:44 rrm0d
crw-rw---- 1 root     daemon    14,  4 Jan 18 08:36 rrm0e
crw------- 1 root     superuse  11,  0 Mar 29 15:42 rrp0a
crw------- 1 root     superuse  11,  1 Mar 29 15:43 rrp0b
crw------- 1 root     superuse  11,  2 Mar 29 15:43 rrp0c
crw------- 1 root     superuse  11,  7 Mar 29 15:43 rrp0h
crw------- 1 root     superuse  14,  0 Mar 29 15:40 rxp0a
crw------- 1 root     superuse  14,  1 Mar 29 15:41 rxp0b
crw------- 1 root     superuse  14,  2 Mar 29 15:41 rxp0c
crw------- 1 root     superuse  14,  3 Mar 29 15:41 rxp0d
crw------- 1 root     superuse  14,  4 Mar 29 15:41 rxp0e
crw------- 1 root     superuse  14,  5 Mar 29 15:41 rxp0f
crw------- 1 root     superuse  14,  6 Mar 29 15:41 rxp0g
crw------- 1 root     superuse  14,  7 Mar 29 15:41 rxp0h
crw-rw-rw- 1 root     superuse  17,  0 Mar 29 15:40 tty
crw--w--w- 1 root     sys       21,  0 Mar  7 09:13 tty00
crw--w--w- 1 root     superuse  21,  1 Mar  7 09:04 tty01
crw--w--w- 1 root     superuse  21,  2 Mar  7 09:04 tty02
crw--w--w- 1 root     superuse  21,  3 Mar  7 09:04 tty03
crw--w--w- 1 root     superuse  21,  4 Jul 26 17:15 tty04
crw--w--w- 1 root     superuse  21,  5 Aug  6 23:09 tty05
crw--w--w- 1 root     superuse  21,  6 Mar  7 09:04 tty06
crw--w--w- 1 root     superuse  21,  7 Mar  7 09:04 tty07
crw------- 1 root     superuse   4,  0 Mar 30 00:14 ttyh0
crw------- 1 root     superuse   4,  1 Mar 30 00:14 ttyh1
crw------- 1 root     superuse   4,  2 Mar 30 00:14 ttyh2
crw------- 1 root     superuse   4,  3 Mar 30 00:14 ttyh3
crw------- 1 root     superuse   4,  4 Mar 30 00:14 ttyh4
crw------- 1 root     superuse   4,  5 Mar 30 00:14 ttyh5
crw------- 1 root     superuse   4,  6 Mar 30 00:14 ttyh6
crw------- 1 root     superuse   4,  7 Mar 30 00:14 ttyh7
crw------- 1 root     superuse   4,  8 Mar 30 00:14 ttyh8
crw------- 1 root     superuse   4,  9 Mar 30 00:14 ttyh9
crw------- 1 root     superuse   4, 10 Mar 30 00:14 ttyha
crw------- 1 root     superuse   4, 11 Mar 30 00:14 ttyhb
crw------- 1 root     superuse   4, 12 Mar 30 00:14 ttyhc
crw------- 1 root     superuse   4, 13 Mar 30 00:14 ttyhd
crw------- 1 root     superuse   4, 14 Mar 30 00:14 ttyhe
crw------- 1 root     superuse   4, 15 Mar 30 00:14 ttyhf
brw------- 1 root     superuse   6,  0 Mar 29 15:40 xp0a
brw------- 1 root     superuse   6,  1 Mar 29 15:40 xp0b
brw------- 1 root     superuse   6,  2 Mar 29 15:41 xp0c
brw------- 1 root     superuse   6,  3 Mar 29 15:41 xp0d
brw------- 1 root     superuse   6,  4 Mar 29 15:41 xp0e
brw------- 1 root     superuse   6,  5 Mar 29 15:41 xp0f
brw------- 1 root     superuse   6,  6 Mar 29 15:41 xp0g
brw------- 1 root     superuse   6,  7 Mar 29 15:41 xp0h
```

Sarebbe disponibile il comando '**shutdown**', ma non sembra funzionare come di consueto. Pertanto, si può usare '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

UNIX # **sync** [*Invio*]

UNIX # [*Ctrl e*]

```
Simulation stopped, PC: 016662 (MOV #200,R4)
```


SIMH sim> **quit** [*Invio*]

Derivazioni di UNIX per hardware ridotto

Dalle versioni dello UNIX di ricerca sono derivate, a suo tempo, delle varianti per sistemi molto poveri di risorse. In particolare, Mini-UNIX e LSX. Si tratta di sistemi in grado di lavorare con una memoria centrale da 64 Kibyte.

Mini-UNIX

Mini-UNIX è un sistema derivato da UNIX versione 6, del quale utilizza lo stesso file system. È importante osservare che, anche se nel file system sono presenti le informazioni sui gruppi di utenti, questi non sono considerati.

Si possono trovare i file-immagine di Mini-UNIX, per dischi RK05, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/Mini-Unix/*. Servono precisamente i file '`tape1.bin.gz`', '`tape2.bin.gz`' e '`tape3.bin.gz`'.

I file vanno estratti e quindi va preparato uno script per SIMH. Nell'estrarre i file gli si attribuisce un nome che sintetizzi il loro contenuto:

GNU/Linux $ **gunzip < tape1.bin.gz > mx_root_rk05.dsk** [*Invio*]

GNU/Linux $ **gunzip < tape2.bin.gz > mx_src_rk05.dsk** [*Invio*]

GNU/Linux $ **gunzip < tape3.bin.gz > mx_man_rk05.dsk** [*Invio*]

Lo script per SIMH può avere il contenuto seguente, nel quale si prevede l'uso di tre dischi:

```
;
; PDP-11/20 (1970)
;
SET     CPU  11/20
SET     CPU  64K
SHOW    CPU
;
; RK05 cartridge disks.
;
SET     RK   ENABLE
;
ATTACH  RK0  mx_root_rk05.dsk
SHOW    RK0
;
ATTACH  RK1  mx_src_rk05.dsk
SHOW    RK1
;
ATTACH  RK2  mx_man_rk05.dsk
SHOW    RK2
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`mx.ini`', si avvia la simulazione così:

GNU/Linux $ **pdp11 mx.ini** [*Invio*]

```
PDP-11 simulator V3.6-1
Disabling CR
Disabling XQ
CPU, 11/20, autoconfiguration on, 64KB
RK0, 1247KW, attached to mx_root_rk05.dsk, write enabled
RK1, 1247KW, attached to mx_src_rk05.dsk, write enabled
RK2, 1247KW, attached to mx_man_rk05.dsk, write enabled
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

SIMH sim> **BOOT RK0** [*Invio*]

Se funziona appare l'invito del settore di avvio ('**@**'), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta di '`rkmx`'.

disk boot **@rkmx** [*Invio*]

```
RESTRICTED RIGHTS

USE, DUPLICATION OR DISCLOSURE IS SUBJECT TO
RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.
```

```
login: root [Invio]
```

```
# [Invio]
```

Si annotano i file di dispositivo presenti:

```
# chdir /dev [Invio]
```

```
# ls -l [Invio]

total 0
crw-rw-rw-  1 root    1,  1 Jan 26  1976 kmem
crw-rw-rw-  1 root    1,  0 Jan 26  1976 mem
crw-rw-rw-  1 root    1,  2 Jan 26  1976 null
brw-rw-rw-  1 root    0,  0 Sep 17 17:30 rk0
brw-rw-rw-  1 root    0,  1 Sep 18 01:53 rk1
crw--w--w-  1 root    0,  0 Sep 18 01:57 tty8
```

Come si può vedere, sono disponibili i file di dispositivo per due soli dischi, mentre nel simulatore ne sono stati previsti tre. Per aggiungere il file di dispositivo del terzo disco si può procedere così:

```
# /etc/mknod rk2 b 0 2 [Invio]
```

È possibile innestare un solo disco alla volta. Per esempio, volendo aggiungere quello dei sorgenti corrispondente al secondo, si può procedere così:

```
# /etc/mount /dev/rk1 /mnt [Invio]
```

Poi, per il distacco del disco si procede come di consueto, specificando il file di dispositivo:

```
# /etc/umount /dev/rk1 [Invio]
```

Non è prevista la procedura di arresto del sistema ed è disponibile solo '**sync**', dopo il quale è possibile interrompere il funzionamento del simulatore con la combinazione [*Ctrl e*].

```
# sync [Invio]
```

```
# [Ctrl e]

Simulation stopped, PC: 016662 (BNE 16674)
```

```
sim> quit [Invio]
```

LSI UNIX o LSX

LSX è un sistema derivato da UNIX versione 6; per microprocessore LSI-11, ridotto al punto di poter funzionare con soli 48 Kibyte di memoria centrale.

Si possono trovare i file-immagine di LSX, per dischetti RX01, presso *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/LSX/* ; precisamente serve il file *http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/LSX/lsximgs.tar.bz2* che contiene tutto il necessario per questi esempi.

I file-immagine vanno estratti e quindi va preparato uno script per SIMH.

```
$ tar xjvf lsximgs.tar.bz2 [Invio]
```

Si ottengono diversi file, tra cui, in particolare, '`root.dsk`' e '`usr.dsk`'. Lo script per SIMH può avere il contenuto seguente, nel quale si prevede l'uso di due dischi:

```
;
; LSI-11 with only 48 Kibyte RAM memory.
;
SET     CPU  48K
SHOW    CPU
;
; RX01 floppy disk.
;
SET     RX   ENABLE
;
ATTACH  RX0  root.dsk
SHOW    RX0
;
ATTACH  RX1  usr.dsk
SHOW    RX1
;
; Should boot manually.
;
```

Se lo script è contenuto nel file '`lsx.ini`', si avvia la simulazione così:

```
$ pdp11 lsx.ini [Invio]

PDP-11 simulator V3.6-1
Disabling CR
CPU, 11/73, NOCIS, autoconfiguration on, 48KB
RX: buffering file in memory
RX0, 256KB, attached to root.dsk, write enabled
RX: buffering file in memory
RX1, 256KB, attached to usr.dsk, write enabled
```

Quindi, dall'invito di SIMH si dà il comando di avvio:

```
sim> BOOT RX0 [Invio]
```

Se funziona appare l'invito del settore di avvio ('**rx boot:**'), dal quale va scritto il nome del file del kernel da eseguire: in questo caso si tratta di '`lsx`'.

```
rx boot:lsx [Invio]
```

Si ottiene subito l'invito della shell. Dal momento che il terminale si presenta configurato per le lettere maiuscole, conviene regolare subito questa cosa:

```
# STTY -LCASE [Invio]
```

Si annotano i file di dispositivo presenti:

```
# chdir /dev [Invio]
```

```
# ls -l [Invio]

total 0
brw-rw-rw- 1 0       0,  0 Jun  8 15:00 fd0
brw-rw-rw- 1 0       0,  1 Oct 29 03:10 fd1
crw-rw-rw- 1 0       0,  0 Jul  1  1977 tty8
```

Nel sistema che si ottiene mancano programmi importanti e anche l'innesto del secondo dischetto può essere impossibile.

```
# sync [Invio]
```

```
# [Ctrl e]

Simulation stopped, PC: 015650 (MOV R3,(SP))
```

```
sim> quit [Invio]
```

Programmi di servizio

Le varie versioni dello UNIX di ricerca utilizzano dei file system inaccessibili con i sistemi attuali. Ciò rende difficile il trasferimento di dati con un file-immagine contenente uno dei vecchi UNIX. Probabilmente, l'unico programma che venga in aiuto per questo è V7fs, che comunque occorre compilare in proprio, ma almeno funziona in un sistema GNU/Linux comune.

V7fs

V7fs è un programma in grado di leggere un file-immagine contenente un file system di UNIX versione 7, di attraversare il suo contenuto e di estrapolare i file. Il programma va raccolto in forma sorgente da *http://minnie.tuhs.org/Archive/PDP-11/Tools/Filesys/v7fs-0.1.tar.gz* . Dopo l'estrazione si ottiene in particolare il file '`v7fs.c`' e il file-make; pertanto si può compilare così:

```
$ make v7fs [Invio]
```

Si ottiene il file eseguibile '`v7fs`' nella directory corrente. Supponendo di disporre del file '`unix_v7_root_rl02.dsk`', contenen-

te un file system da scorrere con V7fs, si può procedere nel modo seguente:

```
GNU/Linux$ ./v7fs unix_v7_rl02.dsk [Invio]
```

V7fs funziona in modo interattivo e mostra un invito, dal quale si possono dare comandi simili a quelli di un vecchio sistema UNIX:

```
V7fs* ? [Invio]

commands:
        ls [-i] [dir]: list directory contents, current dir default
        cd name: change to directory 'name'
        cat name1: print file 'name1' on terminal
        cp name1 [name2]: copy internal file 'name1' to external 'name2'
               name2 defaults to name1.
               (An i-number can be used instead of name1 for cp or cat.)
        cpdir: copy all files in current internal directory
               to current external directory
        lcd name: change to local directory 'name'
        printi ino ...: print contents of inode 'ino'
        printblk blk ...: print contents of block 'blk'
        printsb: print contents of the super block
        dumpblk blk ...: hex dump of block 'blk'
        dumpboot: hex dump of the boot block
        cpblk file blk ...: copy contents of 'blk' to external file 'file'
               (append to file if it exists)
        rootino ino: read directory with inode 'ino', making it
               the root directory
        ! : shell escape; the rest of the line is passed to the shell
        q or ^d: quit


V7fs* ls [Invio]

.               ..              bin             boot
dev             etc             hphtunix        hptmunix
lib             mdec            rkunix          rl2unix
rphtunix        rptmunix        usr

V7fs* cd etc [Invio]

V7fs* ls [Invio]

.               ..              accton          cron
ddate           dmesg           fsck            getty
group           init            mkfs            mknod
mount           mtab            passwd          rc
ttys            umount          update          utmp
wall

V7fs* cat group [Invio]

other::1:
sys::2:bin,sys
bin::3:sys,bin
uucp::4:

V7fs* cp group [Invio]
```

L'ultimo comando mostra la copia del file '`group`' all'esterno, in un file con lo stesso nome, nella directory corrente nel momento dell'avvio del programma.

```
V7fs* q [Invio]
```

Riferimenti

- Bob Supnik, *The Computer History Simulation Project*
  http://simh.trailing-edge.com/
- Robert M. Supnik, *SIMH user's guide*
  http://simh.trailing-edge.com/pdf/simh_doc.pdf
- Robert M. Supnik, *PDP-11 simulator usage*
  http://simh.trailing-edge.com/pdf/pdp11_doc.pdf
- Phil Budne, *Article 1995 of alt.sys.pdp10, How big were they?*
  http://www.inwap.com/pdp10/usenet/disks
- John Holden, *YAPP - Yet another PDP-11 Page*
  http://www.psych.usyd.edu.au/pdp-11/
- Steven Schulz, *Installing and operating 2.11BSD on the PDP-11*, 1995
  http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.pdf

http://minnie.tuhs.org/PUPS/Setup/2.11bs11d_setup.html

- *2.11BSD*
  http://minnie.tuhs.org/Archive/PDP-11/Distributions/ucb/2.11BS11D/
  http://minnie.tuhs.org/Archive/PDP-11/Boot_Images/2.11_on_Simh/
- Warren Toomey, *Details of the PUPS Archive*, 1996; *FAQ on the Unix Archive and Unix on the PDP-11*, 2001; *What Unixes run on What PDPs?*
  http://minnie.tuhs.org/PUPS/archive_details.html
  http://minnie.tuhs.org/PUPS/pupsfaq.html
  http://minnie.tuhs.org/PUPS/node6.html
- John Lions, *Lions' Commentary on Unix 6th Edition with Source Code*, Peer-To-Peer Communications, sesta edizione, 1996, ISBN 1573980137
  http://www.amazon.ca/Lions-Commentary-Unix-Source-Code/dp/1573980137
- *Mini Unix*
  http://minnie.tuhs.org/UnixTree/MiniUnix/

[1] **SIMH** software libero con licenza speciale

[2] Rispetto alla documentazione originale, il file '`file8.tar`' viene inserito in coda al nastro principale, senza bisogno di creare un secondo nastro apposito. Nella realtà ciò non sarebbe possibile, per via della capacità limitata del nastro stesso.

[3] La sigla '**`tm`**' va utilizzata in quanto si tratta di un nastro di un'unità a nastro di tipo TM11; se fosse un nastro TS11, va usata probabilmente la sigla '**`ts`**', come descritto nella documentazione di 2.11BSD.

[4] La partizione '**`a:`**' deve iniziare a partire dal primo settore disponibile del disco, altrimenti non è possibile avviare poi il sistema operativo.

[5] La dimensione viene data in settori e la si cambia in base ai calcoli effettuati precedentemente.

[6] Si osservi che è obbligatorio dare il nome '**`swap`**' alla partizione usata per lo scambio della memoria virtuale.

[7] La partizione '**`b:`**' deve cominciare a partire dal settore successivo a quello della partizione '**`a:`**'. Dal momento che la partizione '**`a:`**' è composta da 325755 settori, contando a partire da zero, l'ultimo settore della prima partizione è il numero 325754, pertanto il successivo, che inizia la partizione '**`b:`**' è il numero 325755.

[8] La sigla '**`tm0`**' fa riferimento alle unità TM11.