Parte xxxv

Manuale di os16

«

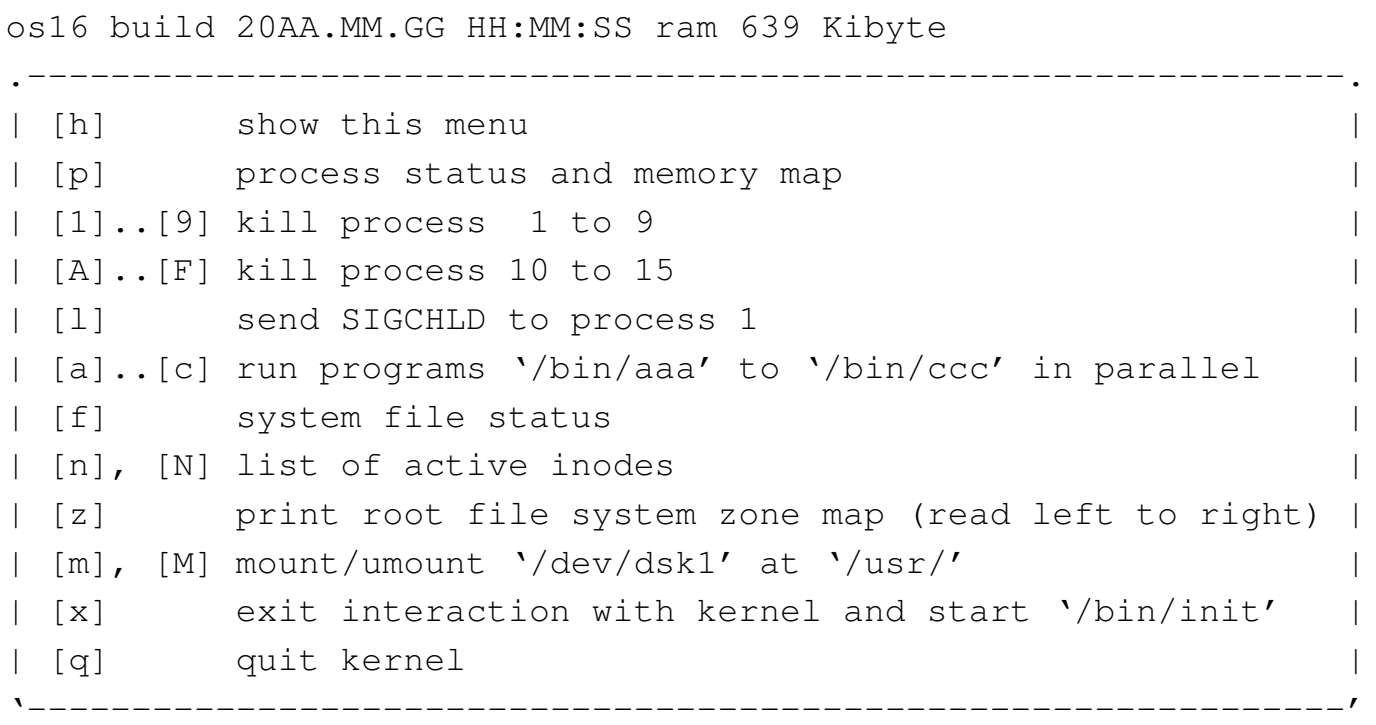
Avvio del sistema e conclusione

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.



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.

Tasto	Risultato
[*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
fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffe00000003fffffffffffffffffffffff
fffe000000000001ffffffff0007ffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff800
00000000000000000000000000000000000000007fffffffffffffffffffffffffffffffffffff
fffffffffffffffffffffffffffffffffffffffff7fffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
fffffffffffbfffffffffffffffffffffffffffffffffffe0000000000000000000000000f

abbaaaaababaaababaaabbaabbaababababbaabbbbbbbbbbbbbb
```

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.

Sezione 1: programmi eseguibili o comandi interni di shell

«

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].

«

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].

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 termina-

le 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]

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].

«

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].

«

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].

«

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].

«

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].

«

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].

«

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].

«

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].

Sezione 3: funzioni di libreria

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 macro-variabile 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].

«

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].

«

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].

«

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].

«

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***, as-

sociandovi 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].

«

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].

«

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.

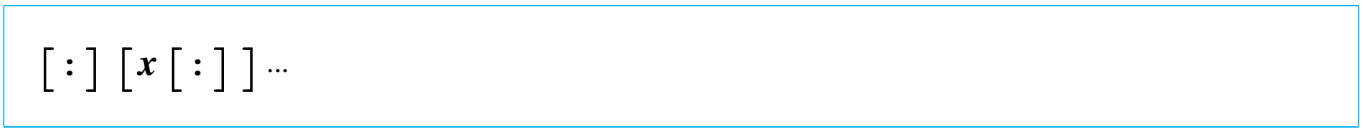
La variabile ***opterr*** serve per configurare il comportamento di ***getopt()***. Questa variabile contiene inizialmente il valore 1. Quando ***getopt()*** incontra un'opzione per la quale si richiede un

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:



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]

os16: getpwent(3)

«

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 fun-

zione 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].

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].

«

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].

«

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].

«

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 lette-

ra 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 in-

terpretare, 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].

«

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].

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.

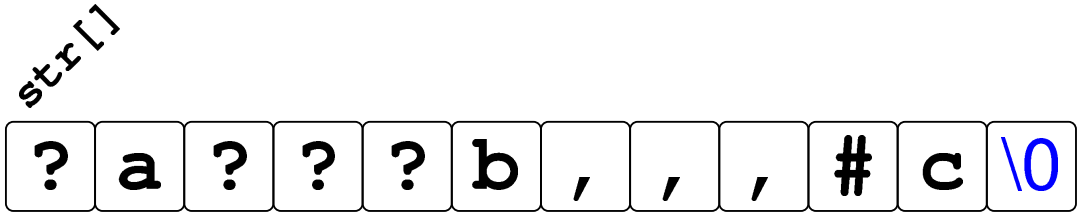
Per comprendere lo scopo della funzione viene utilizzato lo stesso esempio che appare nel documento *ISO/IEC 9899:TC2*, al paragrafo 7.21.5.7, con qualche piccola modifica per poterlo rendere un programma autonomo:

```
#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;
}
```

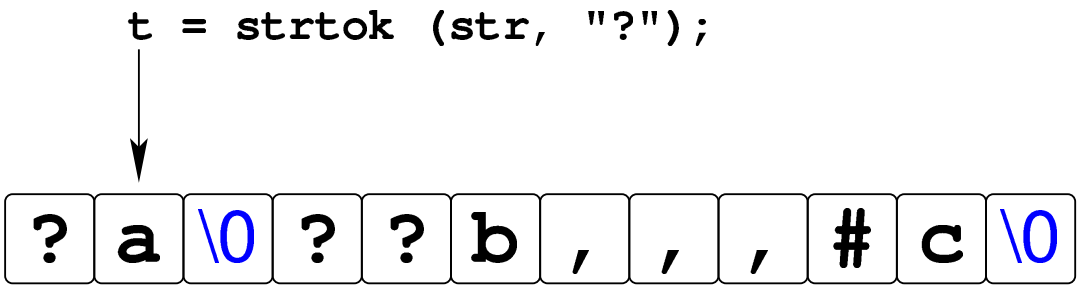
Avviando il programma si ottiene quanto già descritto dai commenti inseriti nel codice:

```
strtok: "a"
strtok: "??b"
strtok: "c"
strtok: "(null)"
```

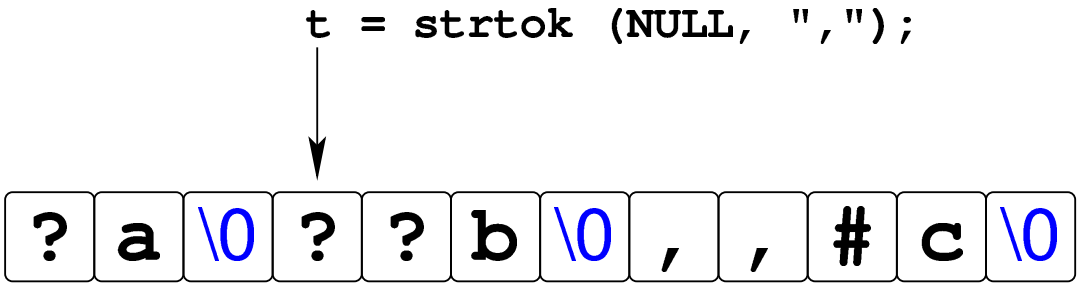
Ciò che avviene nell'esempio può essere schematizzato come segue. Inizialmente la stringa '**str**' ha in memoria l'aspetto seguente:



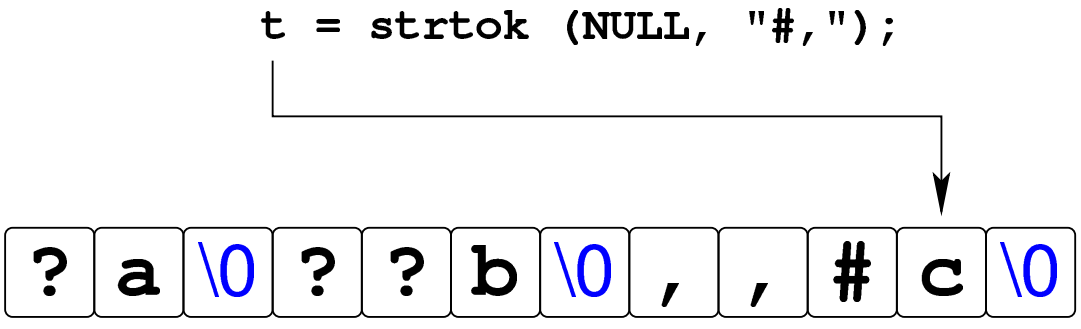
Dopo la prima chiamata della funzione ***strtok()*** la stringa risulta alterata e il puntatore ottenuto raggiunge la lettera '**a**':



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 nu-

merazione 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*** caratte-

ri 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 que-

ste 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]. «

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].

«

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)

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].

«

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].

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)

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].

Sezione 9: kernel

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.

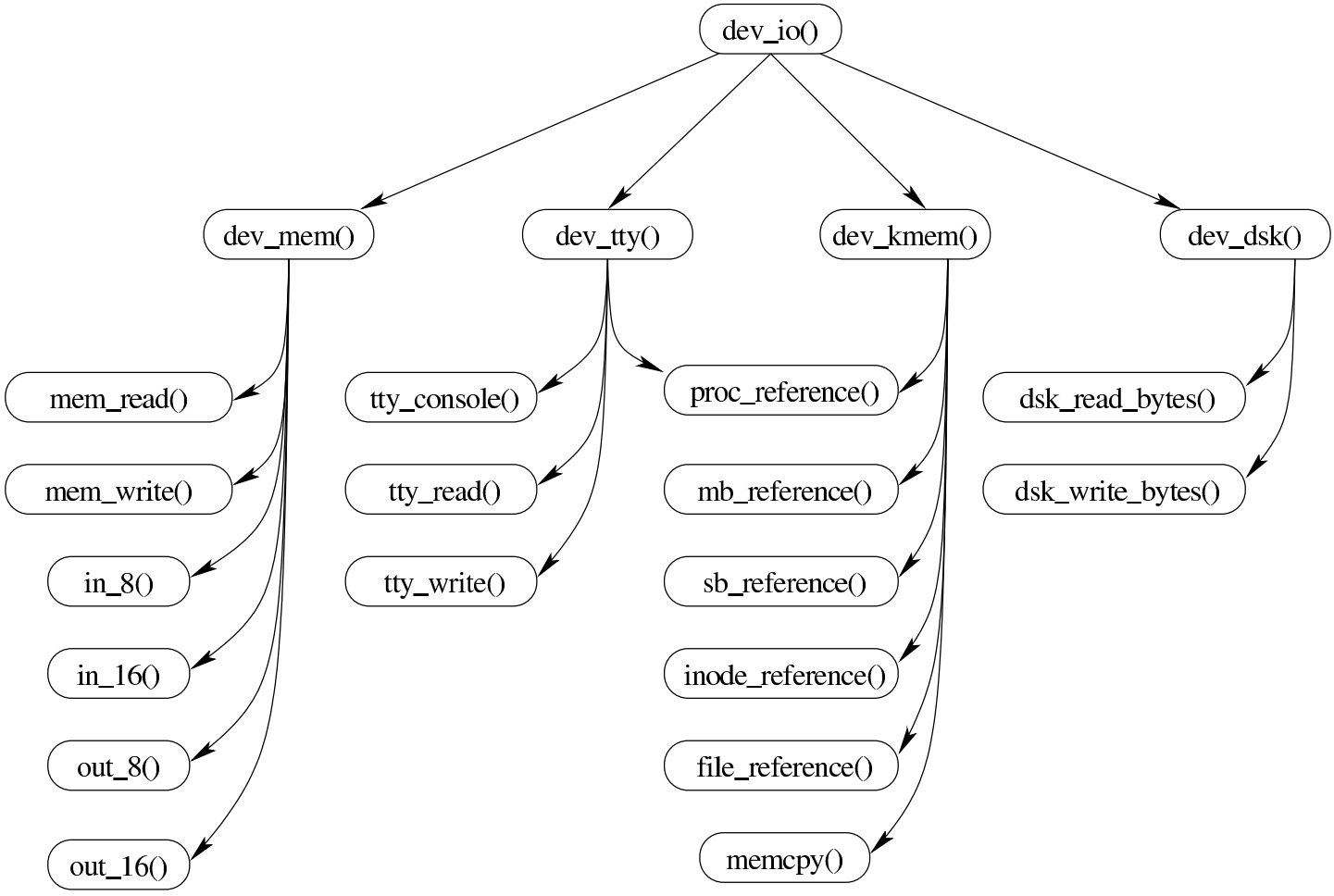
Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_MEM`	r/w	diretto	Permette l'accesso alla memoria, in modo indiscriminato, perché os16 non offre alcun tipo di protezione al riguardo.
`DEV_NULL`	r/w	nessuno	Consente la lettura e la scrittura, ma non si legge e non si 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.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`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.
`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.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`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.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`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].

os16: dev_dsk(9)

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]

os16: dev_kmem(9)

«

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]

os16: dev_mem(9)

«

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.

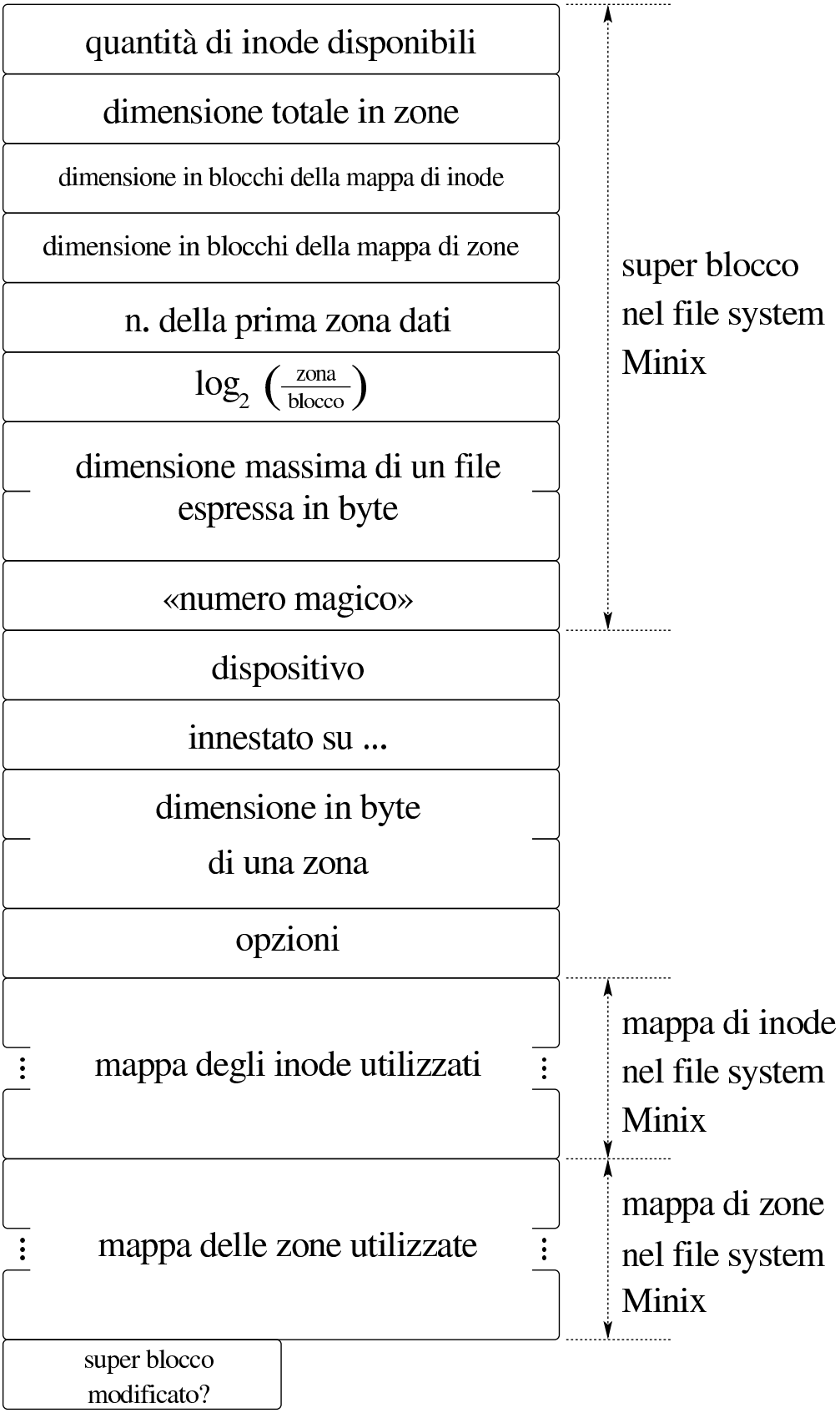
Alcune di queste funzioni sono usate, altre no. Per esempio durante il funzionamento interattivo del kernel vengono usate ***print_proc_list()***, ***print_segments()***, ***print_kmem()***, ***print_time()*** e ***print_mb_map()***.

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.

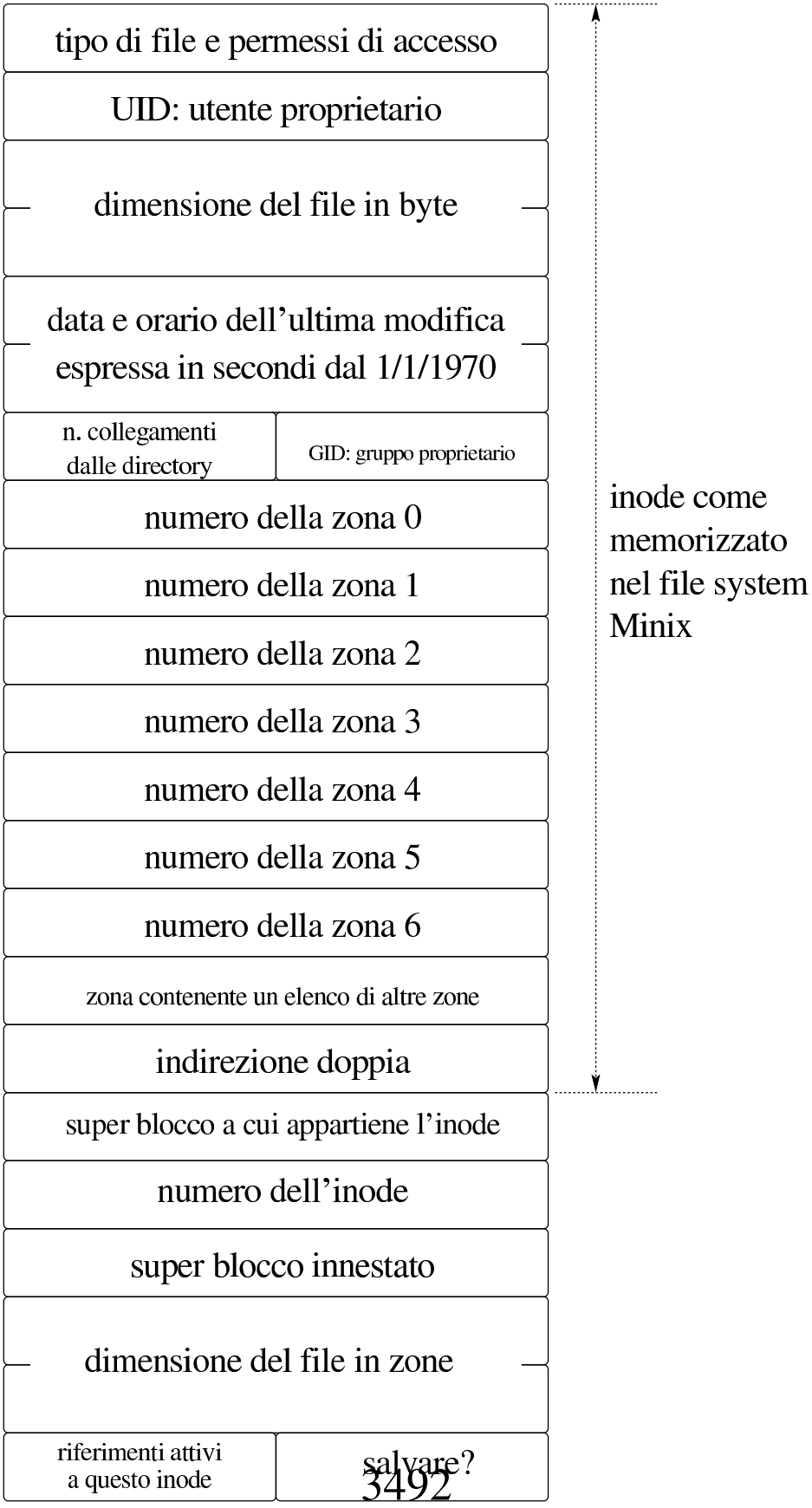
Figura u148.1. Struttura del tipo '**`sb_t`**', corrispondente agli elementi dell'array ***sb_table[]***.



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;
};
```

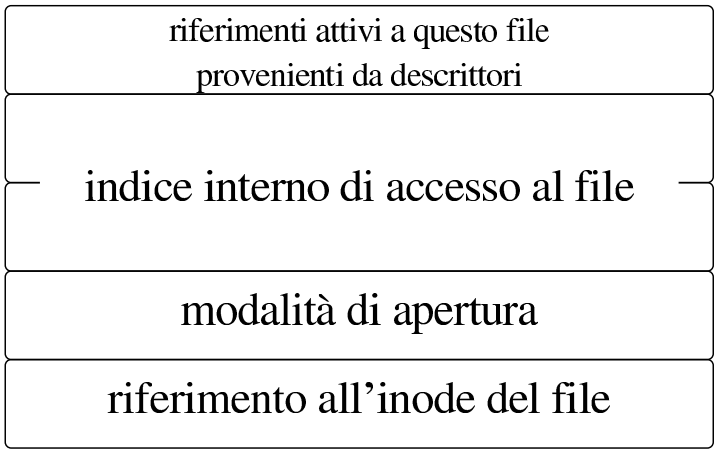
Figura u148.6. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.



Listato u148.7. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.

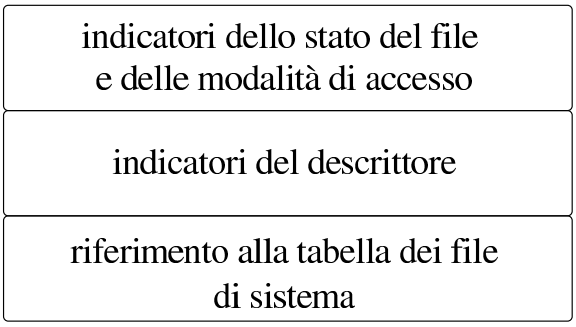
```
<verbatimpre width="60">
<![CDATA[
typedef struct inode     inode_t;

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 ti-

po '**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 opera-

zione 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 macro-

variabili 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].

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].

«

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].

«

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 nel-

la 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].

«

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].

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 sul-

l'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, nel-

l'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].

os16: path_link(9)

«

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].

os16: path_mkdir(9)

«

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].

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].

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].

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, resti-

tuisce 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].

os16: zone_alloc(9)

«

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].

os16: zone_free(9)

Vedere *zone_alloc(9)* [i159.3.51].

os16: zone_read(9)

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()***.

Funzione o macroistruzione	Descrizione

`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()***.

Funzione o macroistruzione	Descrizione
`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()***.

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()***.

Funzione o macroistruzione	Descrizione

`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.

Funzione	Descrizione
`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()***.

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_...()`**'.

Funzione	Descrizione
`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()***.

Funzione	Descrizione
`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/’.

Funzione	Descrizione
`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.

Funzione	Descrizione
`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.

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`**'.
`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.

Membro	Contenuto

`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.

Membro	Contenuto
`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.

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].

os16: proc_available(9)

«

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].

os16: proc_find(9)

«

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]

os16: proc_init(9)

«

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]

«

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]

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].

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].

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].

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].

os16: proc_sys_exec(9)

«

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***.

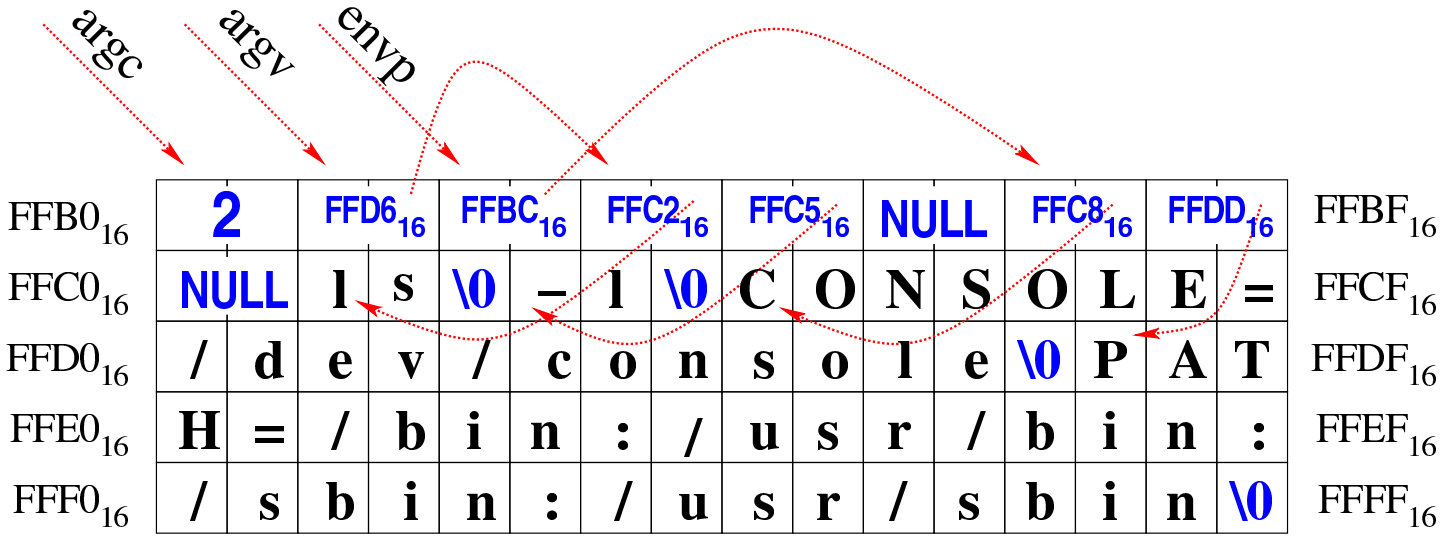
DESCRIZIONE

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].

os16: proc_sys_exit(9)

«

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].

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].

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].

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].

«

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].

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].

«

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].

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.

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.

Funzione	Descrizione
`void tty_write (dev_t` ***device*****`,`** `int` ***c*****`);`**	Scrive sullo schermo del terminale rappresentato dal numero di dispositivo, il carattere fornito come secondo parametro.