Capitolo 10

Processi di elaborazione

10.1 Introduzione ai processi

Un programma singolo, nel momento in cui viene eseguito, è un ***processo***. La nascita di un processo, cioè l'avvio di un programma, può avvenire solo tramite una richiesta da parte di un altro processo già esistente. Si forma quindi una sorta di gerarchia dei processi organizzata ad albero. Il processo principale (*root*) che genera tutti gli altri, è quello dell'eseguibile '**`init`**' che a sua volta è attivato direttamente dal kernel.

In linea di principio, il programma avviato dal kernel come processo principale, può essere qualunque cosa, anche una shell (tenendo conto, comunque, che il kernel predilige l'eseguibile '`/sbin/init`'), ma in tal caso si tratta di applicazioni specifiche e non di un sistema standard.

Qui si preferisce utilizzare il nome Init per identificare il processo principale, sapendo che questo si concretizza generalmente nell'eseguibile '**`init`**'.

10.1.1 Tabella dei processi

«

Il kernel gestisce una tabella dei processi che serve a tenere traccia del loro stato. In particolare sono registrati i valori seguenti:

- il nome dell'eseguibile in funzione;
- gli eventuali argomenti passati all'eseguibile al momento dell'avvio attraverso la riga di comando;
- il numero di identificazione del processo;
- il numero di identificazione del processo che ha generato quello a cui si fa riferimento;
- il nome del dispositivo di comunicazione se il processo è controllato da un terminale;
- il numero di identificazione dell'utente;
- il numero di identificazione del gruppo;

Il kernel Linux rende disponibile i dati della tabella dei processi attraverso un file system virtuale innestato nella directory '`/proc/`'. Dalla presenza di questo file system virtuale dipende la maggior parte dei programmi che si occupano di gestire i processi.

In particolare, a partire da questa directory se ne diramano altre, tante quanti sono i processi in esecuzione, ognuna identificata dal numero del processo stesso. Per esempio, '`/proc/1/`' contiene i file virtuali che rappresentano lo stato del processo numero uno, ovvero Init che è sempre il primo a essere messo in funzione (il processo zero corrisponde al kernel). Il listato seguente mostra il contenuto che potrebbe avere il file '`/proc/1/status`'.

```
Name:   init
State:  S (sleeping)
Pid:    1
PPid:   0
Uid:    0       0       0       0
Gid:    0       0       0       0
Groups:
VmSize:      764 kB
VmLck:         0 kB
VmRSS:        16 kB
VmData:       64 kB
VmStk:         4 kB
VmExe:        24 kB
VmLib:       628 kB
SigPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000057f0d8fc
SigCgt: 00000000280b2603
CapInh: 00000000fffffeff
CapPrm: 00000000ffffffff
CapEff: 00000000fffffeff
```

10.1.2 Nascita e morte di un processo

Come già accennato, la nascita di un processo, cioè l'avvio di un programma, può avvenire solo tramite una richiesta da parte di un altro processo già esistente, utilizzando la chiamata di sistema ***fork()***. Per esempio, quando si avvia un programma attraverso il terminale, è l'interprete dei comandi (la shell) che genera il processo corrispondente.

Quando un processo termina, lo fa attraverso la chiamata di sistema ***exit()***, trasformandosi in un processo «defunto», o «zombie». È poi

il processo che lo ha generato che si deve occupare di eliminarne le tracce.

Il processo genitore, per avviare l'eliminazione dei suoi processi defunti, deve essere avvisato che ne esiste la necessità attraverso un segnale '**SIGCHLD**'. Questo segnale viene inviato proprio dalla funzione di sistema ***exit()***, ma se il meccanismo non funziona come previsto, si può inviare manualmente un segnale '**SIGCHLD**' al processo genitore. In mancanza d'altro, si può far terminare l'esecuzione del processo genitore stesso.

Il processo che termina potrebbe avere avviato a sua volta altri processi (figli). In tal caso, questi vengono affidati al processo numero uno, cioè Init.

A volte, l'interruzione di un processo provoca il cosiddetto ***scarico della memoria*** o *core dump*. In pratica si ottiene un file nella directory corrente, contenente l'immagine del processo interrotto. Per tradizione, questo file è denominato '`core`', in onore dei primo tipo di memoria centrale che sia stato utilizzato con un sistema Unix: la memoria a nuclei magnetici, ovvero *core memory*. Questi file servono a documentare un incidente di funzionamento e a permetterne l'analisi attraverso strumenti diagnostici opportuni. Solitamente questi file possono essere cancellati tranquillamente.

La proliferazione di questi file va tenuta sotto controllo: di solito non ci si rende conto se un processo interrotto ha generato o meno lo scarico della memoria. Ogni tanto vale la pena di fare una ricerca all'interno del file system per rintracciare questi file, come nell'esempio seguente:

```
# find / -name core -type f -print[Invio]
```

Ciò che conta è di non confondere *core* con spazzatura: ci possono essere dei file chiamati ‘`core`’, per qualche motivo, che nulla hanno a che fare con lo scarico della memoria.

10.1.3 Comunicazione tra processi

Nel momento in cui l’attività di un processo dipende da quella di un altro ci deve essere una forma di comunicazione tra i due. Ciò viene definito IPC, o *Inter process communication*, ma questa definizione viene confusa spesso con un tipo particolare di comunicazione definito IPC di System V. I metodi utilizzati normalmente sono di tre tipi: invio di segnali, condotti (flussi di dati FIFO) e IPC di System V.

I ***segnali*** sono dei messaggi elementari che possono essere inviati a un processo, permettendo a questo di essere informato di una condizione particolare che si è manifestata e di potersi uniformare. I programmi possono essere progettati in modo da intercettare questi segnali, allo scopo di compiere alcune operazioni prima di adeguarsi agli ordini ricevuti. Nello stesso modo, un programma potrebbe anche ignorare completamente un segnale, o compiere operazioni diverse da quelle che sarebbero prevedibili per un tipo di segnale determinato. Segue un elenco dei segnali più importanti.

Segnale	Descrizione
SIGINT	È un segnale di interruzione intercettabile, inviato normalmente attraverso la tastiera del terminale, con la combinazione [*Ctrl c*], al processo che si trova a funzionare in primo piano (*foreground*). Di solito, il processo che riceve questo segnale viene interrotto.

Segnale	Descrizione
SIGQUIT	È un segnale di interruzione intercettabile, inviato normalmente attraverso la tastiera del terminale, con la combinazione [*Ctrl* \], al processo che si trova a funzionare in primo piano. Di solito, il processo che riceve questo segnale viene interrotto.
SIGTERM	È un segnale di conclusione intercettabile, inviato normalmente da un altro processo. Di solito, provoca la conclusione del processo che ne è il destinatario.
SIGKILL	È un segnale di interruzione non intercettabile, che provoca la conclusione immediata del processo. Non c'è modo per il processo destinatario di eseguire alcuna operazione di salvataggio o di scarico dei dati.
SIGHUP	È un ***segnale di aggancio*** che rappresenta l'interruzione di una comunicazione. In particolare, quando un utente esegue un *logout*, i processi ancora attivi avviati eventualmente sullo sfondo (*background*) ricevono questo segnale. Può essere generato anche a causa della «morte» del processo controllante.

La tabella 10.3 elenca i segnali descritti dallo standard POSIX, mentre l'elenco completo può essere ottenuto consultando *signal(7)*.

Tabella 10.3. Segnali gestiti dai sistemi GNU/Linux secondo lo standard POSIX.

Segnale	Azione	Descrizione
SIGHUP	A	Il collegamento con il terminale è stato interrotto.
SIGINT	A	Interruzione attraverso un comando dalla tastiera.

Segnale	Azione	Descrizione
SIGQUIT	A	Conclusione attraverso un comando dalla tastiera.
SIGILL	A	Istruzione non valida.
SIGABRT	C	Interruzioni di sistema.
SIGFPE	C	Eccezione in virgola mobile.
SIGKILL	AEF	Conclusione immediata.
SIGSEGV	C	Riferimento non valido a un segmento di memoria.
SIGPIPE	A	Condotto (flusso FIFO) interrotto.
SIGALRM	A	Timer.
SIGTERM	A	Conclusione.
SIGUSR1	A	Primo segnale definibile dall'utente.
SIGUSR2	A	Secondo segnale definibile dall'utente.
SIGCHLD	B	Eliminazione di un processo figlio.
SIGCONT		Riprende l'esecuzione se in precedenza è stato fermato.
SIGTSTOP	DEF	Ferma immediatamente il processo.
SIGTSTP	D	Stop attraverso un comando della tastiera.
SIGTTIN	D	Processo sullo sfondo che richiede dell'input.
SIGTTOU	D	Processo sullo sfondo che deve emettere dell'output.

Le lettere contenute nella seconda colonna rappresentano il comportamento predefinito dei programmi che ricevono tale segnale:

- '**A**' termina il processo;
- '**B**' il segnale viene ignorato;
- '**C**' la memoria viene scaricata (*core dump*);

- '**D**' il processo viene fermato;
- '**E**' il segnale non può essere catturato;
- '**F**' il segnale non può essere ignorato.

L'utente ha a disposizione in particolare due mezzi per inviare segnali ai programmi:

- la combinazione di tasti [*Ctrl c*] che di solito genera l'invio di un segnale '**SIGINT**' al processo in esecuzione sul terminale o sulla console attiva;
- l'uso di '**kill**' (programma o comando interno di shell) per inviare un segnale particolare a un processo stabilito.

Attraverso la shell è possibile collegare più processi tra loro in un ***condotto***, come nell'esempio seguente, in modo che lo standard output di uno sia collegato direttamente con lo standard input del successivo.

```
$ cat mio_file | sort | lpr[Invio]
```

Ogni connessione tra un processo e il successivo, evidenziata dalla barra verticale, si comporta come un serbatoio provvisorio di dati ad accesso FIFO (*First in first out*: il primo a entrare è il primo a uscire).

È possibile creare esplicitamente dei ***serbatoi FIFO*** di questo genere, in modo da poterli gestire senza dover fare ricorso alle funzionalità della shell. Questi, sono dei file speciali definiti proprio «FIFO» e vengono creati attraverso il programma '**mkfifo**'. Nell'esempio seguente viene mostrata una sequenza di comandi con i quali, creando due file FIFO, si può eseguire la stessa operazione indicata nel condotto visto poco sopra.

`$` **`mkfifo fifo1 fifo2`** [*Invio*]

Crea due file FIFO: '`fifo1`' e '`fifo2`'.

`$` **`cat mio_file >> fifo1 &`** [*Invio*]

Invia '`mio_file`' a '`fifo1`' senza attendere ('**`&`**').

`$` **`sort < fifo1 >> fifo2 &`** [*Invio*]

Esegue il riordino di quanto ottenuto da '`fifo1`' e invia il risultato a '`fifo2`' senza attendere ('**`&`**').

`$` **`lpr < fifo2`** [*Invio*]

Accoda la stampa di quanto ottenuto da '`fifo2`'.

I file FIFO, data la loro affinità di funzionamento con i condotti gestiti dalla shell, vengono anche chiamati «*pipe* con nome», contrapponendosi ai condotti normali che a volte vengono detti «*pipe* anonimi».

Quando un processo viene interrotto all'interno di un *condotto* di qualunque tipo, il processo che inviava dati a quello interrotto riceve un segnale '**`SIGPIPE`**' e si interrompe a sua volta. Dall'altra parte, i processi che ricevono dati da un processo che si interrompe, vedono concludersi il flusso di questi dati e terminano la loro esecuzione in modo naturale. Quando questa situazione viene segnalata, si potrebbe ottenere il messaggio *broken pipe*.

L'IPC di System V è un sistema di comunicazione tra processi sofisticato che permette di gestire code di messaggi, semafori e memoria condivisa. Teoricamente un sistema Unix può essere privo di questa gestione, per esempio un kernel Linux può essere compilato senza

questa funzionalità, ma in pratica conviene che sia presente, perché molti programmi ne dipendono. Una delle questioni ricorrenti che riguardano la gestione dell'IPC di System V è la gestione della memoria condivisa. Si accede a queste informazioni con l'aiuto del programma '**sysctl**':

```
# sysctl -a | less [Invio]
```

```
…
kernel.shmmni = 4096
kernel.shmall = 2097152
kernel.shmmax = 33554432
…
```

Generalmente è meglio non toccare questi valori, ma in alcuni documenti si fa riferimento a tale possibilità, per poter utilizzare dei programmi particolarmente pesanti dal punto di vista dell'utilizzo della memoria. A titolo di esempio, volendo raddoppiare il valore della memoria condivisa massima, si potrebbe intervenire così:

```
# sysctl -w kernel.shmmax=67108864 [Invio]
```

Tabella 10.5. Parametri di IPC di System V.

Parametro	Descrizione
SHMMAX	Dimensione massima, in byte, di un segmento di memoria condivisa.
SHMMIN	Dimensione minima di un segmento di memoria condivisa, espresso in byte.
SHMALL	Quantità massima di memoria condivisa che si può usare, espressa byte o in pagine di memoria.
SHMSEG	Quantità massima di segmenti di memoria condivisa per ogni processo elaborativo.

Parametro	Descrizione
SHMMNI	Quantità massima di segmenti di memoria condivisa per tutto il sistema.
SEMMNI	Quantità massima di insiemi di semafori.
SEMMNS	Quantità massima di semafori per tutto il sistema.
SEMMSL	Quantità massima di semafori per insieme.
SEMMAP	Quantità massima di voci nella mappa dei semafori.
SEMVMX	Valore massimo di semaforo.

10.1.4 Scheduling e priorità

La gestione simultanea dei processi è ottenuta normalmente attraverso la suddivisione del tempo di CPU, in maniera tale che a turno ogni processo abbia a disposizione un breve intervallo di tempo di elaborazione. Il modo con cui vengono regolati questi turni è lo *scheduling*, ovvero la pianificazione di questi processi.

La maggiore o minore percentuale di tempo di CPU che può avere un processo è regolata dalla priorità espressa da un numero. Il numero che rappresenta una priorità deve essere visto al contrario di come si è abituati di solito: un valore elevato rappresenta una priorità bassa, cioè meno tempo a disposizione, mentre un valore basso (o negativo) rappresenta una priorità elevata, cioè più tempo a disposizione.[1]

Sotto questo aspetto diventa difficile esprimersi in modo chiaro: una ***bassa priorità*** si riferisce al numero che ne esprime il valore o alle risorse disponibili? Evidentemente si può solo fare attenzione al contesto per capire bene il significato di ciò che si intende.

La priorità di esecuzione di un processo viene definita in modo autonomo da parte del sistema e può essere regolata da parte dell'utente sommandovi il cosiddetto valore ***nice***. Di conseguenza, un va-

lore nice positivo aumenta il valore della priorità, mentre un valore negativo lo diminuisce.

10.1.5 Privilegi dei processi

Nei sistemi operativi Unix c'è la necessità di distinguere i privilegi concessi agli utenti, definendo un nominativo e un numero identificativo riferito all'utente e al gruppo (o ai gruppi) a cui questo appartiene. L'utente fisico è rappresentato virtualmente dai processi che lui stesso mette in esecuzione; pertanto, un'informazione essenziale riferita ai processi è quella che stabilisce l'appartenenza a un utente e a un gruppo. In altri termini, ogni processo porta con sé l'informazione del numero UID e del numero GID, in base ai quali ottiene i privilegi relativi e gli viene concesso o meno di compiere le operazioni per cui è stato avviato.

10.1.6 Variabili di sistema

I sistemi Unix possono includere una funzionalità denominata Sysctl, con la quale è possibile accedere a delle «variabili di sistema», ovvero dei parametri che controllano il comportamento del kernel. Nei sistemi GNU/Linux è possibile realizzare un kernel sprovvisto di tale funzionalità, ma in generale è meglio che sia inclusa.

L'utente '**root**' può accedere alla lettura e alla modifica di queste variabili di sistema attraverso il programma '**sysctl**':

```
sysctl [opzioni]
```

```
sysctl [opzioni] variabile...
```

```
sysctl [opzioni] -w variabile=valore...
```

Per elencare la situazione di tutte le variabili di sistema si usa normalmente l'opzione '**-a**':

```
# sysctl -a[Invio]

  sunrpc.nlm_debug = 0
  sunrpc.nfsd_debug = 0
  sunrpc.nfs_debug = 0
  sunrpc.rpc_debug = 0
  ...
  fs.quota.lookups = 0
  fs.lease-break-time = 45
  fs.dir-notify-enable = 1
  fs.leases-enable = 1
  fs.overflowgid = 65534
  fs.overflowuid = 65534
  fs.dentry-state = 200   13      45      0       0       0
  fs.file-max = 8192
  fs.file-nr = 315        47      8192
  fs.inode-state = 216    6       0       0       0       0       0
  fs.inode-nr = 216       6
```

L'elenco che si ottiene è comunque più lungo di come si vede da questo esempio. Per conoscere in modo particolare lo stato di una o di alcune variabili basta indicare i loro nomi alla fine della riga di comando:

```
# sysctl kernel.shmmax kernel.domainname[Invio]

  kernel.shmmax = 33554432
  kernel.domainname = (none)
```

Per modificare una variabile si usa l'opzione '**-w**':

```
# sysctl -w kernel.shmmax=67108864 [Invio]
```

Generalmente non c'è alcuna necessità di cambiare i valori delle variabili accessibili attraverso Sysctl; tuttavia, se così fosse, potrebbe essere utile fare in modo che tali modifiche vengano ripristinate ogni volta che si riavvia il sistema operativo. Oltre alla possibilità di realizzare uno script eseguito automaticamente in fase di avvio, è possibile intervenire nel file '`/etc/sysctl.conf`', che si compone semplicemente di direttive di assegnamento a variabili che fanno parte di Sysctl:

```
# /etc/sysctl.conf - Configuration file for setting system
#                    variables.
# See sysctl.conf (5) for information.

kernel.domainname = brot.dg
```

L'esempio mostra l'assegnamento alla variabile '**kernel.domainname**' della stringa '**brot.dg**'. Si osservi che gli spazi prima e dopo il valore assegnato vengono ignorati. Come si può intuire, il carattere '**#**' introduce un commento che viene ignorato fino alla fine della riga, così come vengono ignorate le righe vuote e quelle bianche.

È bene ribadire che generalmente non c'è alcun bisogno di intervenire nella modifica delle variabili di sistema controllate attraverso Sysctl, pertanto è normale che sia presente il file '`/etc/sysctl.conf`', ma commentato completamente.

10.2 Procedura di inizializzazione del sistema (System V)

Quando un sistema Unix viene avviato, il kernel si prende cura di avviare il processo iniziale, Init, a partire dal quale vengono poi generati tutti gli altri: di solito si utilizza un meccanismo di inizializzazione derivato dallo UNIX System V. Init determina quali siano i processi da avviare successivamente, in base al contenuto di '`/etc/inittab`' il quale a sua volta fa riferimento a script contenuti normalmente all'interno della directory '`/etc/rc.d/`', '`/etc/init.d/`', o in un'altra analoga. All'interno di '`/etc/inittab`' si distinguono azioni diverse in funzione del ***livello di esecuzione*** (*run level*), di solito un numero da zero a sei. Per convenzione, il livello zero identifica le azioni necessarie per fermare l'attività del sistema, in modo da permetterne lo spegnimento; il livello sei riavvia il sistema; il livello uno mette il sistema in condizione di funzionare in modalità monoutente.

L'organizzazione della procedura di inizializzazione del sistema e dei livelli di esecuzione costituisce il punto su cui si distinguono maggiormente le distribuzioni GNU. Benché alla fine si tratti sempre della stessa cosa, il modo di strutturare e di collocare gli script è molto diverso da una distribuzione all'altra. Quando si acquista più esperienza, ci si accorge che queste differenze non sono poi un grosso problema, ma all'inizio è importante comprendere e accettare che il sistema operativo e la sua distribuzione particolare mostra un'interpretazione della soluzione del problema e non il risultato definitivo.

10.2.1 Init

«

Init è il processo principale che genera tutti gli altri. All'avvio del sistema legge il file '`/etc/inittab`' il quale contiene le informazioni per attivare gli altri processi necessari, compresa la gestione dei terminali. Per prima cosa viene determinato il livello di esecuzione iniziale, ottenendo l'informazione dalla direttiva '**`initdefault`**' di '`/etc/inittab`'. Quindi vengono attivati i processi essenziali al funzionamento del sistema e infine i processi che combaciano con il livello di esecuzione attivato.

L'eseguibile '**`init`**' può essere invocato dall'utente '**`root`**' durante il funzionamento del sistema, per cambiare il livello di esecuzione, oppure per ottenere il riesame del suo file di configurazione ('`/etc/inittab`'):

```
init [opzioni]
```

Tabella 10.9. Alcune opzioni.

Opzione	Descrizione
`-t` ***secondi***	Stabilisce il numero di secondi di attesa prima di cambiare il livello di esecuzione. In mancanza si intende 20 secondi.
`0`|`1`|`2`|`3`|`4`|`5`|`6`	Un numero da zero a sei stabilisce il livello di esecuzione a cui si vuole passare.
`a`|`b`|`c`	Una lettera '**`a`**', '**`b`**' o '**`c`**' richiede di eseguire soltanto i processi indicati all'interno di '`/etc/inittab`' che hanno un livello di esecuzione pari alla lettera specificata. In pratica, una lettera non indica un livello di esecuzione vero e proprio, in quanto si tratta di una possibilità di configurazione del file '`/etc/inittab`' per definire i cosiddetti livelli «a richiesta» (*on demand*).
`Q`|`q`	Richiede di riesaminare il file '`/etc/inittab`' (dopo che questo è stato modificato).
`S`|`s`	Richiede di passare alla modalità monoutente, ma non è pensato per essere utilizzato direttamente, in quanto per questo si preferisce selezionare il livello di esecuzione numero uno.

Segue la descrizione di alcuni esempi.

- `#` **`init 1`** [*Invio*]

  Pone il sistema al livello di esecuzione uno: monoutente.

- `#` **`init 0`** [*Invio*]

  Pone il sistema al livello di esecuzione zero: arresto del

sistema. Equivale (in linea di massima) all'esecuzione di '**shutdown -h now**'.

- `# init 6` [*Invio*]

  Pone il sistema al livello di esecuzione sei: riavvio. Equivale (in linea di massima) all'esecuzione di '**shutdown -r now**'.

10.2.2 File di configurazione «/etc/inittab»

Il file '`inittab`' descrive quali processi vengono avviati al momento dell'avvio del sistema e durante il funzionamento normale di questo. Init, il processo principale, distingue diversi livelli di esecuzione, per ognuno dei quali può essere stabilito un gruppo diverso di processi da avviare.

La struttura dei record (intesi come righe divise in campi) che compongono le direttive di questo file, può essere schematizzata nel modo seguente:

id : ***livelli_di_esecuzione*** : ***azione*** : ***processo***

I campi vanno interpretati così:

- ***id*** è una sequenza unica di due caratteri che identifica il record (la riga) all'interno di '**inittab**';
- ***livelli_di_esecuzione*** elenca i livelli di esecuzione con cui l'azione indicata deve essere eseguita;
- ***azione*** indica l'azione da eseguire;
- ***processo*** specifica il processo da eseguire.

Se il nome del processo inizia con un simbolo '**+**', Init non esegue l'aggiornamento di '`/var/run/utmp`' e '`/var/log/wtmp`' per quel processo; ciò è utile quando il processo stesso provvede da solo a questa operazione (la descrizione del significato e dell'importanza di questi due file si trova nella sezione 16.2).

Il penultimo campo dei record di questo file, identifica l'azione da compiere. Questa viene rappresentata attraverso una parola chiave, come descritto dall'elenco seguente.

Parola chiave	Descrizione
`respawn`	Quando il processo termina, viene riavviato.
`wait`	Il processo viene avviato una volta (sempre che il livello di esecuzione lo consenta) e Init attende che termini prima di eseguirne degli altri.
`once`	Il processo viene eseguito una volta quando il livello di esecuzione lo consente.
`boot`	Il processo viene eseguito al momento dell'avvio del sistema. Il campo del livello di esecuzione viene ignorato.
`bootwait`	Il processo viene eseguito al momento dell'avvio del sistema e Init attende la fine del processo prima di proseguire. Il campo del livello di esecuzione viene ignorato.
`off`	Non fa alcunché.

Parola chiave	Descrizione
`ondemand`	Si tratta dei record «a richiesta» che vengono presi in considerazione quando viene richiamato Init seguito da una lettera '**a**', '**b**' o '**c**', che rappresentano appunto tre possibili livelli di esecuzione *on demand*. Le lettere '**a**', '**b**' o '**c**' non sono livelli di esecuzione, ma solo un modo per selezionare i processi *on demand* indicati all'interno del file '`/etc/inittab`'.
`initdefault`	Permette di definire il livello di esecuzione predefinito per l'avvio del sistema. Se non viene specificato, Init richiede l'inserimento di questo valore attraverso la console.
`sysinit`	Il processo viene eseguito al momento dell'avvio del sistema, prima di quelli indicati come '**boot**' e '**bootwait**'. Il campo del livello di esecuzione viene ignorato.
`powerwait`	Il processo viene eseguito quando Init riceve il segnale '**SIGPWR**' che indica un problema con l'alimentazione elettrica. Init attende la fine del processo prima di proseguire.
`powerfail`	Il processo viene eseguito quando Init riceve il segnale '**SIGPWR**' che indica un problema con l'alimentazione elettrica. Init **non** attende la fine del processo.
`powerokwait`	Il processo viene eseguito quando Init ha ricevuto il segnale '**SIGPWR**', che indica un problema con l'alimentazione elettrica, ed è anche presente il file '`/etc/powerstatus`' contenente la parola '**OK**'. Ciò significa che l'alimentazione elettrica è tornata allo stato di normalità.

Parola chiave	Descrizione
`ctrlaltdel`	Il processo viene eseguito quando Init riceve il segnale '**`SIGINT`**'. Ciò significa che è stata premuta la combinazione di tasti [*Ctrl Alt Canc*] ([*Ctrl Alt Del*] nelle tastiere inglesi).
`kbrequest`	Il processo viene eseguito quando Init riceve un segnale dal gestore della tastiera che sta a indicare la pressione di una combinazione speciale di tasti sulla tastiera della console.

Il secondo campo, quello dei livelli di esecuzione, può contenere diversi caratteri che stanno a indicare diversi livelli di esecuzione possibili. Per esempio, la stringa '**`123`**' indica che il processo specificato va eseguito indifferentemente per tutti i livelli di esecuzione da uno a tre. Questo campo può contenere anche una lettera dell'alfabeto: '**`a`**', '**`b`**' o '**`c`**' che sta a indicare un livello a richiesta. Nel caso di azioni del tipo '**`sysinit`**', '**`boot`**' e '**`bootwait`**', il campo del livello di esecuzione viene ignorato.

Negli esempi seguenti, si mostra prima un record del file '`/etc/inittab`' e quindi, sotto, la sua descrizione.

- ```
  id:5:initdefault:
  ```
  Definisce il livello di esecuzione iniziale: cinque.

- ```
  si::sysinit:/etc/rc.d/rc.sysinit
  ```
  Inizializzazione del sistema: è la prima cosa a essere eseguita dopo l'avvio del sistema stesso. In pratica viene avviato lo script '`/etc/rc.d/rc.sysinit`' (Red Hat).

- ```
  l1:1:wait:/etc/rc.d/rc 1
  ```
  ```

Indica di eseguire '`/etc/rc.d/rc`', con l'argomento '**`1`**', nel caso in cui il livello di esecuzione sia pari a uno: singolo utente (Red Hat)

- `rc:2345:wait:/etc/rc.d/rc.M`

  Indica lo script ('`/etc/rc.d/rc.M`') da eseguire per i livelli di esecuzione da due a cinque (Slackware).

- `ca::ctrlaltdel:/sbin/shutdown -t5 -rfn now`

  Indica il programma da eseguire in caso di pressione della combinazione [*Ctrl Alt Canc*]. Il livello di esecuzione non viene indicato perché è indifferente (Slackware).

- `l0:0:wait:/etc/rc.d/rc 0`

  Indica di eseguire '`/etc/rc.d/rc`', con l'argomento '**`0`**', nel caso in cui il livello di esecuzione sia pari a zero: arresto del sistema (Red Hat).

- `l6:6:wait:/etc/rc.d/rc 6`

  Indica di eseguire '`/etc/rc.d/rc`', con l'argomento '**`6`**', nel caso in cui il livello di esecuzione sia pari a sei: riavvio (Red Hat).

- `pf::powerfail:/sbin/shutdown -f +5 "THE POWER IS FAILING"`

  Indica il programma da eseguire quando si verifica un problema con l'alimentazione elettrica (una vecchia edizione Slackware).

- `pf::powerfail:/sbin/genpowerfail start"`

  Come nell'esempio precedente, ma realizzato in modo differente (Slackware).

- `pg:0123456:powerokwait:/sbin/shutdown -c "THE POWER IS BACK"`

Indica il programma da eseguire se l'alimentazione elettrica torna normale prima del completamento del processo avviato quando si è verificato il problema (una vecchia edizione Slackware).

- `pg::powerokwait:/sbin/genpowerfail stop"`

  Come nell'esempio precedente, ma realizzato in modo differente (Slackware).

- ```
  1:12345:respawn:/sbin/mingetty tty1
  2:2345:respawn:/sbin/mingetty tty2
  3:2345:respawn:/sbin/mingetty tty3
  4:2345:respawn:/sbin/mingetty tty4
  5:2345:respawn:/sbin/mingetty tty5
  6:2345:respawn:/sbin/mingetty tty6
  ```

  Si tratta dell'elenco di console virtuali utilizzabili. La prima si attiva per tutti i livelli di esecuzione da uno a cinque, le altre solo per i livelli superiori a uno. In questo caso è '**`mingetty`**' a essere responsabile dell'attivazione delle console virtuali (Red Hat).

- `s1:45:respawn:/sbin/agetty -L ttyS0 9600 vt100`

  Indica l'attivazione di un terminale connesso sulla prima porta seriale. Si attiva solo con i livelli di esecuzione quattro o cinque (Slackware).

- `d2:45:respawn:/sbin/agetty -mt60 38400,19200,9600,2400,1200 ttyS1 vt100`

  Indica l'attivazione di un terminale remoto connesso via modem sulla seconda porta seriale. Si attiva solo con i livelli di esecuzione quattro o cinque (Slackware).

- `x:5:respawn:/usr/bin/X11/xdm -nodaemon`

  Nel caso il livello di esecuzione sia pari a cinque, esegue '`/usr/bin/X11/xdm`' che si occupa di avviare una procedura di accesso (*login*) all'interno dell'ambiente grafico X (Red Hat).
  ```

10.2.3 Script «/etc/initscript»

«

Quando lo script di shell ‘`/etc/initscript`’ esiste, viene utilizzato da Init per avviare i processi indicati all’interno del file ‘`/etc/inittab`’.

```
/etc/initscript id livello_di_esecuzione azione processo
```

Di solito questo script non è presente, tuttavia potrebbe essere utile per definire delle variabili di ambiente e altre impostazioni che riguardano l’interpretazione degli script della procedura di inizializzazione del sistema. La documentazione *initscript(5)* mostra un esempio simile a quello seguente, che dovrebbe chiarire il senso di questa possibilità.

```
# initscript   Executed by init(8) for every program it
#              wants to spawn like this:
# /bin/sh /etc/initscript <id> <level> <action> <process>

# Set umask to safe level, and enable core dumps.
umask 022
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
# Execute the program.
eval exec "$4"
```

Come si vede anche dai commenti dell’esempio, ‘**`initscript`**’ riceve da Init degli argomenti che rappresentano tutti i campi contenuti nel record corrispondente di ‘`/etc/inittab`’.

10.2.4 Procedura di attivazione e disattivazione dei servizi

La prima differenza importante che distingue le varie distribuzioni GNU sta nell'organizzazione degli script della procedura di inizializzazione del sistema. Il punto di riferimento comune è Init con il suo '`/etc/inittab`'; da questo file si intende quali siano il comandi avviati in presenza di un livello di esecuzione determinato; quello che c'è dopo costituisce il problema più grosso.

Volendo semplificare molto le cose, si può pensare al fatto che ci dovrebbe essere una directory specifica, contenente un gruppetto di script utilizzato esclusivamente per questi scopi. La directory in questione non ha una collocazione standard, salvo il fatto che storicamente si trova nell'ambito della gerarchia '`/etc/`'. Le motivazioni che spingono a un'impostazione differente di questi script della procedura di inizializzazione del sistema, possono essere varie. Anche la collocazione di tale directory è controversa, a cominciare dal fatto che la directory '`/etc/`' non dovrebbe contenere programmi e nemmeno script.[2]

Gli script della procedura di inizializzazione del sistema hanno il compito di avviare il sistema operativo e di fermarlo, attivando e disattivando tutti i servizi necessari, cioè intervenendo nell'avvio e nella conclusione del funzionamento dei demoni relativi.

Si può intuire che non sia possibile realizzare uno o più script del genere per avviare tutti i tipi di demone che possono essere presenti nel proprio sistema, anche perché ci possono essere dei servizi installati che però non si vogliono gestire. Di conseguenza, nella situazione più banale, quando si intende installare e gestire un nuovo servizio, occorre anche modificare la procedura di inizializzazione del siste-

ma per attivare il demone relativo e per disattivarlo nel momento dell'arresto del sistema. Una cosa del genere può andare bene per una persona esperta, ma si tratta sempre di un'impostazione piuttosto scomoda.

Secondo una convenzione diffusa, per facilitare l'avvio e la conclusione dei servizi si definisce una directory specifica, che potrebbe essere '`/etc/rc.d/init.d/`', o '`/etc/init.d/`', o ancora '`/sbin/init.d/`', all'interno della quale si possono inserire degli script che hanno una sintassi uniforme.

```
nome_servizio {start|stop}
```

In pratica, il nome dello script tende a corrispondere a quello del servizio che si intende controllare; l'argomento costituito dalla parola chiave '**`start`**' fa sì che lo script avvii il servizio, mentre la parola chiave '**`stop`**' serve a concluderlo.

Questi script possono essere più o meno raffinati, per esempio possono accettare anche altri tipi di ordini (come '**`restart`**', allo scopo di riavviare un servizio), ma la cosa più importante è che dovrebbero evitare di avviare dei doppioni, controllando prima di avviare qualcosa, se per caso questo risulta già attivo. Naturalmente, un servizio può essere ottenuto con l'avvio di diversi programmi demone e in questo è molto comodo tale sistema di script specifici.

A titolo di esempio viene mostrato come potrebbe essere composto uno script del genere, per l'avvio del servizio ipotetico denominato '**`pippo`**', che si avvale del programma omonimo per gestirlo. Per semplicità, non vengono indicati accorgimenti particolari per controllare che il servizio sia già attivo o meno.

```
#!/bin/sh
# init.d/pippo {start|stop|restart}

# Analisi dell'argomento usato nella chiamata.
case "$1" in
  start)
        printf "Avvio del servizio Pippo: "
        /usr/sbin/pippo &
        printf "\n"
        ;;
  stop)
        printf "Disattivazione del servizio Pippo: "
        killall pippo
        printf "\n"
        ;;
  restart)
        killall -HUP pippo
        ;;
  *)
        echo "Utilizzo: pippo {start|stop|restart}"
        exit 1
esac

exit 0
```

Lo scopo e la vera utilità di questi script sta nel facilitare una standardizzazione della procedura di inizializzazione del sistema; tuttavia si può intuire la possibilità di sfruttarli anche per attivare e disattivare manualmente un servizio, senza intervenire direttamente sui programmi relativi.

Procedendo intuitivamente, si potrebbe pensare di fare in modo che la procedura di inizializzazione del sistema, provveda a eseguire tut-

ti gli script di controllo dei servizi, utilizzando l'argomento '**`start`**' all'avvio e l'argomento '**`stop`**' allo spegnimento. Una cosa del genere è molto semplice da realizzare, ma si pongono due problemi: alcuni servizi potrebbero essere a disposizione, senza che la procedura di inizializzazione del sistema debba avviarli automaticamente; inoltre la sequenza di attivazione e di disattivazione dei servizi potrebbe essere importante.

In pratica, si utilizza un meccanismo molto semplice: si predispongono tante directory quanti sono i livelli di esecuzione gestiti attraverso il file '`/etc/inittab`'. Queste directory hanno il nome '`rc`***n***`.d/`', dove ***n*** rappresenta il numero del livello di esecuzione corrispondente. La loro collocazione effettiva potrebbe essere '`/etc/rc`***n***`.d/`', '`/etc/rc.d/rc`***n***`.d/`' o anche '`/sbin/init.d/rc`***n***`.d/`'. All'interno di queste directory si inseriscono dei collegamenti simbolici che puntano agli script descritti nella sezione precedente, in modo che siano presenti i riferimenti ai servizi desiderati per ogni livello di esecuzione (distinto in base alla directory '`rc`***n***`.d/`' particolare).

I nomi di questi collegamenti iniziano in modo speciale: '`K`***nn***' e '`S`***nn***'. I collegamenti che iniziano con la lettera «S» (*Start*) servono per individuare gli script da utilizzare per l'attivazione dei servizi, vengono avviati con l'argomento '**`start`**', in ordine alfabetico, in base alla sequenza fissata con le due cifre numeriche successive che servono proprio a distinguerne la sequenza. I collegamenti che iniziano con la lettera «K» (*Kill*) servono per individuare gli script da utilizzare per la disattivazione dei servizi, vengono avviati con l'argomento '**`stop`**', anche questi in ordine alfabetico. Ecco cosa potrebbe contenere una di queste directory:

```
$ tree rc6.d[Invio]

rc6.d/
|-- K11cron -> ../init.d/cron
|-- K14ppp -> ../init.d/ppp
|-- K15fetchmail -> ../init.d/fetchmail
|-- K19aumix -> ../init.d/aumix
|-- K19setserial -> ../init.d/setserial
|-- K20boa -> ../init.d/boa
|-- K20exim -> ../init.d/exim
|-- K20gpm -> ../init.d/gpm
|-- K20inetd -> ../init.d/inetd
|-- K20lprng -> ../init.d/lprng
|-- K20makedev -> ../init.d/makedev
|-- K20pcmcia -> ../init.d/pcmcia
|-- K20postgresql -> ../init.d/postgresql
|-- K20psad -> ../init.d/psad
|-- K20ssh -> ../init.d/ssh
|-- K25hwclock.sh -> ../init.d/hwclock.sh
|-- K30etc-setserial -> ../init.d/etc-setserial
|-- K79nfs-common -> ../init.d/nfs-common
|-- K80nfs-kernel-server -> ../init.d/nfs-kernel-server
|-- K85bind9 -> ../init.d/bind9
|-- K89atd -> ../init.d/atd
|-- K89hotplug -> ../init.d/hotplug
|-- K89klogd -> ../init.d/klogd
|-- K90sysklogd -> ../init.d/sysklogd
|-- S10portmap -> ../init.d/portmap
|-- S19devfsd -> ../init.d/devfsd
|-- S20sendsigs -> ../init.d/sendsigs
|-- S30urandom -> ../init.d/urandom
|-- S31umountnfs.sh -> ../init.d/umountnfs.sh
|-- S35networking -> ../init.d/networking
|-- S40umountfs -> ../init.d/umountfs
|-- S50raid2 -> ../init.d/raid2
```

```
`-- S90reboot -> ../init.d/reboot

0 directories, 37 files
```

A titolo di esempio viene mostrato un pezzo di uno script, per una shell Bourne o derivata, fatto per scandire un elenco di collegamenti del genere, allo scopo di attivare e di disattivare i servizi, a partire dai collegamenti contenuti nella directory '`/etc/rc3.d/`'. Per un lettore inesperto, questo potrebbe essere un po' difficile da leggere, ma l'esempio viene aggiunto per completare l'argomento.

```
#!/bin/sh
…
…
# Attivazione dei servizi del livello di esecuzione 3.
for I in /etc/rc3.d/K*;
do
    # Disattiva il servizio.
    $I stop
done
#
for I in /etc/rc3.d/S*;
do
    # Attiva il servizio.
    $I start
done
```

In pratica, prima si disattivano i servizi corrispondenti ai collegamenti che iniziano con la lettera «K», quindi si attivano quelli che hanno la lettera «S». Si può intuire che le directory '`rc0.d/`' e '`rc6.d/`' contengano prevalentemente, o esclusivamente, riferimenti che iniziano con la lettera «K», dal momento che i livel-

li di esecuzione corrispondenti portano all'arresto del sistema o al suo riavvio.

10.3 Situazione dei processi

Le informazioni sulla situazione dei processi vengono ottenute a partire dalla tabella dei processi messa a disposizione dal kernel. Dal momento che il meccanismo attraverso cui queste informazioni possono essere ottenute dal kernel non è standardizzato per tutti i sistemi Unix, questi programmi che ne permettono la consultazione hanno raramente un funzionamento conforme.

Il meccanismo utilizzato in particolare dal kernel Linux è quello del file system virtuale innestato nella directory '`/proc/`'. A questo proposito, è il caso di osservare che il pacchetto dei programmi di servizio che permettono di conoscere lo stato dei processi è denominato Procps, in riferimento a questa particolarità del kernel Linux.

10.3.1 Process status

Il controllo dello stato dei processi esistenti avviene fondamentalmente attraverso l'uso di '**ps**',[3] '**pstree**'[4] e '**top**'.[5] Il primo mostra un elenco di processi e delle loro caratteristiche, il secondo un albero che rappresenta la dipendenza gerarchica dei processi e il terzo l'evolversi dello stato di questi.

I programmi '**ps**' e '**pstree**' rappresentano la situazione di un istante: il primo si presta per eventuali rielaborazioni successive, mentre il secondo è particolarmente adatto a seguire l'evoluzione di una catena di processi, specialmente quando a un certo punto si verifica una transizione nella proprietà dello stesso (UID).

\# **ps** [*Invio*]

```
  PID TTY STAT  TIME COMMAND
  374   1 S    0:01 /bin/login -- root
  375   2 S    0:00 /sbin/mingetty tty2
  376   3 S    0:00 /sbin/mingetty tty3
  377   4 S    0:00 /sbin/mingetty tty4
  380   5 S    0:00 /sbin/mingetty tty5
  382   1 S    0:00 -bash
  444  p0 S    0:00 su
  445  p0 S    0:00 bash
  588  p0 R    0:00 ps
```

```
$ pstree -u -p[Invio]
```

```
init(1)-+-atd(868,daemon)
        |-bdflush(6)
        |-boa(728,www-data)
        |-cron(871)
        |-devfsd(40)
        |-diskmond(812)
        |-getty(879)
        |-getty(882)
        …
        |-sh(881,tizio)---startx(889)---xinit(900)-+-Xorg(901,root)
        |                                          `-xinitrc(905)---fvwm2(907)
        …
        `-xinetd(857)
```

Invece, il programma '**top**' impegna un terminale (o una finestra di terminale all'interno del sistema grafico) per mostrare costantemente l'aggiornamento della situazione. Si tratta quindi di un controllo continuo, con l'aggiunta però della possibilità di interferire con i processi inviandovi dei segnali o cambiandone il valore nice.

Figura 10.32. Il programma '**top**'.

```
 10:13pm  up 58 min,  5 users,  load average: 0.09, 0.03, 0.01
67 processes: 65 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  5.9% user,  0.7% system,  0.0% nice, 93.5% idle
Mem:   62296K av, 60752K us,   1544K free, 36856K sh, 22024K buff
Swap: 104416K av,     8K us, 104408K free             16656K cach

PID USER   PRI NI SIZE  RSS SHARE STAT LIB %CPU %MEM TIME COMMAND
588 root    16  0 6520 6520  1368 R      0  5.1 10.4 0:02 X
613 daniele  6  0  736  736   560 R      0  1.3  1.1 0:00 top
596 daniele  1  0 1108 1108   872 S      0  0.1  1.7 0:00 fvwm2
  1 root     0  0  388  388   336 S      0  0.0  0.6 0:08 init
  2 root     0  0    0    0     0 SW     0  0.0  0.0 0:00 kflushd
  3 root     0  0    0    0     0 SW     0  0.0  0.0 0:00 kswapd
 82 root     0  0  352  352   300 S      0  0.0  0.5 0:00 kerneld
139 root     0  0  448  448   364 S      0  0.0  0.7 0:00 syslogd
148 root     0  0  432  432   320 S      0  0.0  0.6 0:00 klogd
159 daemon   0  0  416  416   340 S      0  0.0  0.6 0:00 atd
```

I programmi che visualizzano la situazione dei processi, utilizzano spesso delle sigle per identificare alcune caratteristiche. La tabella 10.33 ne descrive alcune.

Tabella 10.33. Elenco di alcune delle sigle utilizzate dai programmi che permettono di consultare lo stato dei processi in esecuzione.

Sigla	Descrizione
UID	Il numero UID dell'utente proprietario del processo.
PID	Il numero del processo, cioè il PID.
PPID	Il numero PID del processo genitore (quello da cui ha avuto origine).
USER	Il nome dell'utente proprietario del processo.
PRI	La priorità del processo.
NI	Il valore nice.
SIZE	La dimensione complessiva dell'immagine del processo.

Sigla	Descrizione
RSS	La dimensione della porzione del processo residente effettivamente nella memoria centrale.
SWAP	La dimensione della porzione del processo non residente nella memoria centrale (che pertanto si trova in una memoria di scambio o *swap*).
SHARE	La quantità di memoria condivisa utilizzata dal processo.
WCHAN	L'evento per cui il processo è in attesa.
STAT	Lo stato del processo.
TT	Il terminale, se il processo ne utilizza uno.
TIME	Il tempo totale di utilizzo della CPU.
CTIME	Il tempo di CPU sommando anche l'utilizzo da parte dei processi figli.
COMMAND	Il comando utilizzato per avviare il processo.

In particolare, lo stato del processo rappresentato dalla sigla '**STAT**', viene descritto da una o più lettere alfabetiche il cui significato viene riassunto nella tabella 10.34.

Tabella 10.34. Lo stato del processo espresso attraverso una o più lettere alfabetiche.

Lettera	Stato
`R`	In funzione (residente in memoria).
`S`	In pausa o dormiente.
`D`	In pausa non interrompibile.
`T`	Sospeso.

Lettera	Stato
`Z`	Defunto (zombie).
`X`	Morto.
`W`	Non utilizza la memoria centrale (pertanto è spostato completamente in una memoria di scambio).
`N`	Ha un valore nice positivo (in pratica è rallentato).

Tabella 10.35. Assieme allo stato del processo potrebbero apparire altri simboli che aggiungono informazioni.

Simbolo	Stato
`<`	Processo con un valore nice minore di zero.
`N`	Processo con un valore nice maggiore di zero.
`L`	Ha delle porzioni bloccate in memoria.
`s`	Processo principale di una sessione (di solito si tratta della shell con cui si avviano altri processi).
`+`	Processo in primo piano.

10.3.2 Utilizzo di «ps»

Il programma '**`ps`**' (*Process status*) visualizza un elenco dei processi in corso di esecuzione. Se non viene specificato diversamente, si ottiene solo l'elenco dei processi che appartengono all'utente.

`ps` [***opzioni***] [`pid…`]

Dopo le opzioni possono essere indicati esplicitamente i processi (in forma dei numeri PID) in modo da ridurre a loro l'elenco ottenuto.

Tabella 10.36. Elenco di alcune delle chiavi di ordinamento utilizzabili con l'opzione '**o**', oppure '**--sort**' di '**ps**'.

Chiave	Chiave	Descrizione
c	cmd	Nome dell'eseguibile.
C	cmdline	Riga di comando completa.
o	session	Numero di sessione.
p	pid	PID.
P	ppid	PPID.
r	rss	RSS (memoria residente utilizzata).
t	tty	Terminale.
T	start_time	Orario di inizio del processo.
U	uid	UID.
u	user	Nominativo dell'utente
y	priority	Priorità.

Segue la descrizione di alcune opzioni. Si osservi che le opzioni rappresentate da un carattere singolo, possono iniziare o meno con un trattino: il comando '**ps**' per la tradizione BSD non prevede il trattino davanti alle opzioni, mentre lo standard POSIX lo richiede. Va

però osservato che solo alcune volte le opzioni BSD corrispondono a quelle POSIX

Tabella 10.37. Alcune opzioni.

Opzione	Descrizione

`l`
 	
`-l`
	Emette un elenco lungo, composto in sostanza da più elementi informativi.

`u`	Formato utente: viene indicato in particolare l'utente a cui appartiene ogni processo e l'ora di inizio in cui il processo è stato avviato.
`f`	Visualizza la dipendenza gerarchica tra i processi in modo semplificato.
`a`	Visualizza anche i processi appartenenti agli altri utenti.
`r`	Emette l'elenco dei soli processi in esecuzione effettivamente, escludendo così quelli che per qualunque motivo sono in uno stato di pausa.
`h`	Elimina l'intestazione dall'elenco. Può essere utile quando si vuole elaborare in qualche modo l'elenco.
`t`***x*** `-t`***x***	Permette di ottenere l'elenco dei processi associati al terminale ***x***. Per identificare un terminale, si può utilizzare il nome del file di dispositivo corrispondente, senza il percorso precedente ('`/dev/`').
`e`	Mostra l'ambiente particolare del processo dopo la riga di comando.

Opzione	Descrizione
`w`	Se la riga è troppo lunga consente la visualizzazione di una riga in più: l'opzione può essere indicata più volte in modo da specificare quante righe aggiuntive possono essere utilizzate.
`o` [`+` | `-`] ***chiave*** ↩ ↪ [[`+` | `-`] ***chiave***] ... `--sort=` [`+` | `-`] ***chiave*** ↩ ↪ [`,` [`+` | `-`] ***chiave***] ...	Permette di ottenere un risultato ordinato in base alle chiavi di ordinamento specificate. Le chiavi di ordinamento sono composte da una sola lettera nel caso si usi l'opzione '**`o`**', mentre sono rappresentate da una parola nel caso dell'opzione '**`--sort`**' della versione GNU. Il segno '**`+`**' (sottinteso) indica un ordinamento crescente, mentre il segno '**`-`**' indica un ordinamento decrescente. Le chiavi di ordinamento sono indicate simbolicamente in base all'elenco (parziale) visibile nella tabella 10.36.

Segue la descrizione di alcuni esempi.

- `$` **`ps`** [*Invio*]

  Elenca i processi appartenenti all'utente che dà il comando.

- `$` **`ps a l`** [*Invio*]

  Elenca tutti i processi utilizzando un formato più ampio in modo da fornire più dettagli sui processi.

- `$` **`ps a r`** [*Invio*]

  Elenca tutti i processi in funzione escludendo quelli in pausa.

- `$` **`ps a l OUr`** [*Invio*]

  Elenca tutti i processi in formato allargato e riordinato per UID (numero utente) e quindi in base alla dimensione residente in memoria dei processi.

- `$` **`ps a l --sort=uid,rss`** [*Invio*]

  Equivalente all'esempio precedente.

10.3.3 Utilizzo di «pstree»

Il programma '**`pstree`**' (>*Process tree*) visualizza uno schema ad albero dei processi in corso di esecuzione. È possibile specificare un numero di processo (PID), oppure il nome di un utente per limitare l'analisi.

```
pstree [opzioni] [PID | utente]
```

Di solito, quando da uno stesso genitore si diramano diversi processi con lo stesso nome, questi vengono raggruppati. Per cui, l'esempio seguente rappresenta un gruppo di quattro processi '**`getty`**', tutti discendenti da Init:

`$` **`pstree`** [*Invio*]

```
init-+-…
     …
     |-4*[getty]
     …
```

Tabella 10.39. Alcune opzioni.

Opzione	Descrizione
`-a`	Mostra tutta la riga di comando e non solo il nome del processo.
`-c`	Disabilita l'aggregazione dei processi con lo stesso nome, derivanti dallo stesso genitore.
`-h`	Evidenzia il processo corrente e i suoi predecessori (antenati), se il terminale consente una qualche forma di evidenziazione.
`-l`	Visualizza senza troncare le righe troppo lunghe.
`-p`	Mostra i PID.
`-u`	Mostra la transizione degli UID, quando da un genitore appartenente a un certo utente, viene generato un processo che appartiene a un altro.

Segue la descrizione di alcuni esempi.

- `$` **`pstree -a`** [*Invio*]

  Mostra l'albero dei processi elaborativi, con la riga di comando usata per avviarli. In questo caso, i processi multipli non vengono raccolti assieme.

```
init
  |-acpid -c /etc/acpi/events -s /var/run/acpid.socket
  |-atd
 …
  |-dhcpd3 -q eth0
 …
  `-ypbind
      |-{ypbind}
      `-{ypbind}
```

- `$` **`pstree -p`**[*Invio*]

  Mostra l'albero dei processi elaborativi, con l'indicazione del numero PID per ognuno.

```
init(1)-+-acpid(6342)
        |-atd(7377)
       …
        |-dhcpd3(5725)
       …
        `-ypbind(6246)-+-{ypbind}(6247)
                       `-{ypbind}(6248)
```

- `$` **`pstree -u`**[*Invio*]

  Mostra l'albero dei processi elaborativi, con l'indicazione del cambiamento di utente per conto del quale sono avviati.

```
init-+-acpid
     |-atd(daemon)
    …
     |-dhcpd3
    …
     |-login-sh(tizio)-+-ssh
     |                 `-startx-xinit-+-Xorg(root)
     |                                `-sh-+-fvwm2-+-FvwmTaskBar
     |                                     |       |-…
     |                                     |       `-…
     |                                     `-tee
    …
     `-ypbind-2*[{ypbind}]
```

  Dal risultato, si intende che i processi senza indicazione del nome dell'utente, sono avviati con gli stessi privilegi del processo che a sua volta li genera. Per quanto riguarda '**`init`**', si intende che stia funzionando con i privilegi dell'utente '**`root`**'. Dall'esempio, si vede che il processo denominato '**`Xorg`**', ottiene i privilegi del-

l'utente '**root**', pur essendo avviato da un processo che ha solo i privilegi dell'utente '**tizio**', quindi deve essere stato avviato con i «permessi» SUID-root.

10.3.4 Utilizzo di «top»

Il programma '**top**' visualizza la situazione sull'utilizzo delle risorse di sistema attraverso una tabella dell'attività principale della CPU, cioè dei processi che la impegnano maggiormente.

```
top [opzioni]
```

Lo schema viene aggiornato a brevi intervalli, di conseguenza, impegna un terminale. Durante il suo funzionamento, '**top**' accetta dei comandi espressi con un carattere singolo.

Tabella 10.43. Alcune opzioni.

Opzione	Descrizione
`-d` ***secondi***	Permette di specificare l'intervallo di tempo in secondi che viene lasciato trascorrere tra un aggiornamento e l'altro della tabella. Se non viene indicato questo argomento, l'intervallo di tempo tra gli aggiornamenti della tabella è di cinque secondi.
`-s`	Disabilita la possibilità di utilizzare alcuni comandi in modo interattivo. Può essere utile quando si vuole lasciare funzionare '**top**' in un terminale separato evitando incidenti.
`-i`	Permette di visualizzare anche i processi inattivi o defunti (zombie).
`-c`	Permette di visualizzare la riga di comando, invece del solo nome del programma.

Il programma '**top**' accetta dei comandi interattivi, espressi da un carattere singolo, descritti nella tabella successiva.

Tabella 10.44. Alcuni comandi interattivi.

Comando interattivo	Descrizione
`h` `?`	La lettera '**h**' o il simbolo '**?**' fanno apparire un breve riassunto dei comandi e lo stato delle modalità di funzionamento.
`k`	Permette di inviare un segnale a un processo che viene indicato successivamente. Se il segnale non viene specificato, viene inviato '**SIGTERM**'.
`i`	Abilita o disabilita la visualizzazione dei processi inattivi e dei processi defunti (zombie).
`n` `#`	Cambia la quantità di processi da visualizzare. Il numero che esprime questa quantità viene richiesto successivamente. Il valore predefinito di questa quantità è zero, che corrisponde al numero massimo in base alle righe a disposizione sullo schermo (o sulla finestra) del terminale.
`q`	Termina l'esecuzione di '**top**'.
`r`	Permette di modificare il valore nice di un processo determinato. Dopo l'inserimento della lettera '**r**', viene richiesto il numero PID del processo su cui agire e il valore nice. Un valore nice positivo peggiora le prestazioni di esecuzione di un processo, mentre un valore negativo, che però può essere attribuito solo dall'utente '**root**', migliora le prestazioni. Se non viene specificato il valore nice, si intende 10.

Comando interattivo	Descrizione
`S`	Attiva o disattiva la modalità di visualizzazione cumulativa, con la quale, la statistica sull'utilizzo di risorse da parte di ogni processo, tiene conto anche di quello dei processi figli.
`f` `F`	Permette di aggiungere o eliminare alcuni campi nella tabella dei processi.

10.3.5 Utilizzo di «htop»

«

Il programma '**`htop`**'[6] visualizza la situazione sull'utilizzo delle risorse di sistema, in modo simile a '**`top`**', ma offrendo la possibilità di scorrere l'elenco di tutti i processi, utilizzando comandi interattivi più comodi:

```
htop
```

Inizialmente, '**`htop`**' si presenta così:

`$` **`htop`** [*Invio*]

```
  CPU[||                                1.3%]
  Mem[||||||||||||||||||||||||||263/438MB]
  Swp[||                           0/15123MB]

   PID USER      PR  NI  VIRT   RES   SHR S CPU% MEM% COMM
 3749 Debian-c  16   0  1872    68  1700 S  0.0  0.0 less -Pwless /var/log/syslo
 3790 Debian-c  16   0  1872    68  1700 S  0.0  0.0 less -Pwless /var/log/exim/
13319 root      15   0  6188  1800  5748 S  0.0  0.2 sshd: daniele@pts/140
12768 root      15   0  4848  1780  4028 S  0.0  0.2 SCREEN
13543 daniele   16   0  4656  1704  4264 S  0.0  0.2 /bin/sh
 3900 root      16   0  156M 62648  3072 S  0.0  5.8 /usr/bin/perl -w /usr/sbin/
11305 root      26  10  3712  1980  1624 S  0.0  0.2 /usr/sbin/mathopd -f /etc/m
 3591 root      16   0  1584   356  1416 S  0.0  0.0 /sbin/syslogd -rm 0
 3594 root      16   0  2728   180  1368 S  0.0  0.0 /sbin/klogd -c 3
 3905 root      16   0  1508    32  1356 S  0.0  0.0 /usr/sbin/kmsgsd
    1 root      16   0  1528    80  1376 S  0.0  0.0 init [2]
   32 root      15   0     0     0     0 S  0.0  0.0 kapmd
1Help   2Follow 3Search 4Invert 5Sort<- 6Sort-> 7Nice - 8Nice + 9Kill  10Quit
```

Attraverso i tasti [*freccia−su*] e [*freccia−giù*] è possibile scorrere l'elenco dei processi elaborativi presenti, mentre con la [*barra−spaziatrice*] è possibile selezionare uno o più processi. Nella parte inferiore dello schermo, vengono riepilogati i comandi principali che si impartiscono attraverso l'uso dei primi dieci tasti funzionali.

Tabella 10.46. Comandi principali.

Tasto	Descrizione
[*barra−spaziatrice*]	Seleziona il processo che si trova sotto alla barra di scorrimento. Alcuni comandi possono operare su gruppi di processi; in tal caso si fa riferimento a quelli selezionati in questo modo.
[*F1*], [*h*]	Mostra una guida riassuntiva dei comandi disponibili.

Tasto	Descrizione
[*F2*]	Fa in modo di mantenere in evidenza il processo su cui si trova la barra di scorrimento. Se la barra viene spostata, la richiesta viene annullata.
[*F3*], [*/*]	Esegue una ricerca.
[*F4*]	Inverte l'ordinamento.
[*F5*], [*F6*]	Cambia il criterio di ordinamento in base alla colonna o a quella successiva. L'intestazione della colonna secondo cui è applicato l'ordinamento risulta evidenziata rispetto alle altre.
[*F7*], [*]*], [*F8*], [*[*]	Incrementa o decrementa la priorità di esecuzione del processo evidenziato dalla barra di scorrimento. Solo l'utente '**root**' può aumentare la priorità (ovvero può ridurre il valore *nice*).
[*F9*], [*k*]	Invia un segnale ai processi selezionati, oppure a quello evidenziato dalla barra di selezione. Alla pressione del tasto segue un menù di segnali, tra cui scegliere quello desiderato.
[*F10*], [*q*]	Termina il funzionamento del programma.
[*M*]	Seleziona un ordinamento in base all'utilizzo della memoria.
[*P*]	Seleziona un ordinamento in base all'utilizzo della CPU.
[*C*]	Richiama la selezione delle colonne da visualizzare.

10.3.6 Determinazione del numero PID

Attraverso il programma '**pidof**'[7] è possibile determinare i numeri dei processi elaborativi PID corrispondenti al nome che viene fornito:

```
pidof [opzioni] programma...
```

Per esempio, per conoscere i numeri PID dei processi avviati con il nome '**named**', si usa il comando seguente:

```
$ pidof named[Invio]

  2707
```

Bisogna però considerare che non sempre si ottengono effettivamente tutti i numeri PID; nel caso dell'esempio mostrato la situazione reale potrebbe essere quella seguente:

```
$ pstree -p[Invio]

  init(1)-+-...
          ...
           |-named(2707)-+-{named}(2708)
           |             |-{named}(2709)
           |             `-{named}(2710)
          ...
```

Si veda anche la pagina di manuale *pidof(8)*.

10.4 Accesso ai file

A volte è importante conoscere se un file è utilizzato da qualche processo. Per questo si possono utilizzare i programmi Fuser[8] e Lsof,[9]

che sono in grado di dare qualche informazione aggiuntiva del modo in cui tale file viene utilizzato.

10.4.1 Fuser

Fuser si compone in pratica dell'eseguibile '**fuser**' che si utilizza con la sintassi seguente:

```
fuser [opzioni] file...
```

Il compito normale di Fuser è quello di elencare i processi che utilizzano i file indicati come argomento. In alternativa, '**fuser**' permette anche di inviare un segnale ai processi che utilizzano un gruppo di file determinato, con l'opzione '**-k**'.

L'eseguibile '**fuser**' potrebbe trovarsi nella directory '`/usr/sbin/`', ma può essere utilizzato anche dagli utenti comuni per buona parte delle sue funzionalità.

Quando si utilizza Fuser per ottenere l'elenco dei processi che accedono a file determinati, i numeri di questi processi sono abbinati a una lettera che indica il modo in cui accedono:

Lettera	Descrizione
`c`	directory corrente;
`e`	processo in esecuzione;
`f`	file aperto (spesso questa lettera non viene mostrata affatto);
`F`	file aperto in scrittura (spesso questa lettera non viene mostrata affatto);
`r`	directory radice;

Lettera	Descrizione
`m`	file mappato in memoria o libreria condivisa.

L'eseguibile '**`fuser`**' restituisce il valore zero quando tra i file indicati come argomento ne esiste almeno uno che risulta utilizzato da un processo.

Tabella 10.50. Alcune opzioni.

Opzione	Descrizione
`-a`	Mostra tutti i file indicati nell'argomento, anche se non sono utilizzati da alcun processo. Normalmente, '**`fuser`**' mostra solo i file in uso.
`-k`	Invia un segnale ai processi. Se non viene specificato diversamente attraverso l'opzione '***-segnale***', si utilizza il segnale '**`SIGKILL`**'.
-segnale	Permette di specificare il segnale da inviare con l'opzione '**`-k`**'. In pratica, si tratta di un trattino seguito dal segnale espresso in forma numerica o in forma simbolica (per esempio '**`-TERM`**').
`-l`	Elenca i nomi dei segnali conosciuti.
`-c` `-m`	Utilizzando questa opzione può essere indicato solo un nome di file, il quale può essere un file di dispositivo, riferito a un'unità di memorizzazione innestata nel file system, o una directory che costituisce il punto di innesto della stessa. Quello che si ottiene è l'indicazione di tutti i processi che accedono a quella unità di memorizzazione.
`-u`	Viene aggiunta l'indicazione dell'utente proprietario di ogni processo.
`-v`	Mostra una tabellina dei processi abbinati ai file, in forma più chiara rispetto alla visualizzazione normale.

Opzione	Descrizione
`-s`	Disabilita qualunque emissione di informazioni. Viene utilizzato quando tutto ciò che conta è il solo valore restituito dal programma.

Segue la descrizione di alcuni esempi.

- `#` **`fuser *`** [*Invio*]

  Mostra i processi che accedono ai file della directory corrente.

- `#` **`fuser -k /usr/games/*`** [*Invio*]

  Elimina tutti i processi che utilizzano file nella directory '`/usr/games/`'.

- `#` **`fuser -v -c /dev/sdb1`** [*Invio*]

  Mostra i processi che utilizzano in qualche modo il contenuto dell'unità di memorizzazione '`/dev/sdb1`'. L'uso dell'opzione '**`-v`**' fa sì che si ottengano informazioni dettagliate.

- `#` **`fuser -v -c /mnt`** [*Invio*]

  Mostra i processi che utilizzano in qualche modo il contenuto della directory '`/mnt/`', intesa come punto di innesto per un'unità di memorizzazione esterna.

- `#` **`fuser -k -c /dev/sdb1`** [*Invio*]

  Elimina tutti i processi che utilizzano file nel'unità '`/dev/sdb1`'.

Uno script può utilizzare '**`fuser`**' nel modo seguente per verificare che un file non sia utilizzato da alcun processo prima di eseguire una qualche azione su di esso.

```
#!/bin/sh
MIO_FILE=./mio_file
if fuser -s $MIO_FILE
then
    echo "Il file $MIO_FILE è in uso";
else
    # Esegue qualche azione sullo stesso.
    …
fi
```

10.4.2 Lsof

Lsof serve a elencare i file aperti e si utilizza con la sintassi seguente:

```
lsof [opzioni] [file]…
```

Come si può vedere dal modello, con Lsof non è obbligatoria l'indicazione di un file o di una directory, perché in mancanza di queste informazioni, viene mostrato un elenco completo di file e directory aperte. Questa caratteristica di Lsof facilita la ricerca di file aperti all'interno di una certa posizione della gerarchia del file system (probabilmente scorrendo l'elenco dei file con l'aiuto di '**less**'), quando si cerca di eseguire il distacco di un disco e non si riesce perché un programma lo sta utilizzando.

Segue la descrizione di alcuni esempi.

- `$` **lsof .** [*Invio*]

  Elenca i file della directory corrente che sono aperti da processi appartenenti allo stesso utente che lancia il comando.

- `$` **lsof** [*Invio*]

Elenca tutti i file aperti dal sistema operativo, indipendentemente dai privilegi.

Per approfondire l'uso di Lsof, si può leggere la pagina di manuale *lsof(8)*.

10.5 Informazioni riepilogative

«

Oltre alle informazioni dettagliate sui processi possono essere interessanti delle informazioni riassuntive dell'uso delle risorse di sistema. Si tratta principalmente dell'utilizzo della CPU e della memoria centrale.

È il caso di ricordare che nei sistemi operativi multiprogrammati la CPU esegue i vari processi elaborativi a turno, per piccoli intervalli di tempo, ma i processi possono trovarsi in attesa di poter ricevere input o di poter emettere output, al di fuori della competenza diretta della CPU. Pertanto, la CPU può risultare inutilizzata, anche per la maggior parte del tempo di funzionamento.

Per ottenere queste informazioni si usano in particolare '**uptime**'[10] e '**free**'.[11] Il primo permette di conoscere da quanto tempo è in funzione il sistema senza interruzioni e con quale carico medio, il secondo mostra l'utilizzo della memoria.

```
$ uptime [Invio]

   5:10pm  up  2:21, 6 users, load average: 0.45, 0.48, 0.41
$ free [Invio]
```

```
             total       used       free     shared    buffers     cached
Mem:         22724      22340        384      13884       3664       5600
-/+ buffers:            13076       9648
Swap:        16628       6248      10380
```

10.5.1 Utilizzo di «uptime»

«

```
uptime [opzioni]
```

Emette una sola riga contenente:

- l'orario attuale;
- da quanto tempo è in funzione il sistema;
- quante sessioni di lavoro sono aperte (viene indicato il numero di utenti che risultano collegati al sistema, ma può trattarsi anche dello stesso utente che si collega più volte);
- il carico medio di sistema dell'ultimo minuto, degli ultimi cinque minuti e degli ultimi 15 minuti, espresso in quantità di CPU impiegate.

Di solito, il carico medio è l'informazione meno comprensibile di tutte le altre. Questo valore rappresenta la quantità media di processi attivi, in coda per l'esecuzione da parte del kernel. Per processi attivi, qui si intendono quelli che non sono in pausa per qualche ragione, come l'attesa del completamento di un'altra funzione. Pertanto, un valore inferiore alla quantità di CPU disponibili, indica che la coda dei processi del kernel è rimasta vuota durante parte del tempo preso in considerazione, mentre un valore superiore a tale quantità indica

un certo intasamento, cosa che può diventare preoccupante quando l'unità di tempo presa in considerazione è quella più grande.

```
$ uptime [Invio]

  17:44:33 up  1:47, 4 users, load average: 0.46, 0.71, 1.19
```

In questo esempio si vede che negli ultimi 15 minuti è stato impiegato l'equivalente di 1,19 CPU. Trattandosi di un elaboratore provvisto di una CPU doppia, è come se uno dei due nuclei della CPU fosse rimasto libero l'81 % del tempo.

10.5.2 Utilizzo di «free»

Il programma '**free**' emette attraverso lo standard output delle informazioni relative alla memoria reale e virtuale (*swap*).

```
free [opzioni]
```

Tabella 10.55. Alcune opzioni.

Opzione	Descrizione
`-b`	I valori vengono espressi in byte.
`-k`	I valori vengono espressi in kibibyte (simbolo: «Kibyte») e si tratta della modalità predefinita.
`-t`	Visualizza anche una riga contenente i totali.
`-o`	Disabilita il cosiddetto aggiustamento dei *buffer*. Normalmente, senza questa opzione, la memoria tampone, ovvero quella destinata ai *buffer*, viene considerata libera.

Opzione	Descrizione
`-s` ***secondi***	Permette di ottenere un aggiornamento continuo a intervalli regolari stabiliti dal numero di secondi indicato come argomento. Questo numero può essere anche decimale.

10.6 Controllo diagnostico con Strace

Alle volte può essere utile un controllo maggiore su ciò che fa un programma durante il suo funzionamento. Per questo viene in aiuto Strace[12] (ovvero *system call trace*) che consente di avviare un altro comando e di controllarne le chiamate di sistema e l'uso dei segnali.

Strace si utilizza in pratica attraverso l'eseguibile '**`strace`**', secondo uno dei due modelli sintattici seguenti:

```
strace [opzioni] comando [opzioni_del_comando]
```

```
strace [opzioni] -p pid_da_controllare
```

Le opzioni a disposizione dell'eseguibile '**`strace`**' sono numerose, ma la più importante da ricordare è '**`-o`**', con la quale si specifica il file all'interno del quale inserire le informazioni ottenute durante il funzionamento del comando che viene avviato. Si osservi l'esempio seguente:

```
$ strace -o /tmp/ls.strace ls[Invio]
```

Come si può intendere, si vuole vedere cosa succede avviando il programma '**`ls`**' senza argomenti. Il file '`/tmp/ls.strace`' che si ottiene potrebbe essere simile all'estratto seguente:

```
execve("/bin/ls", ["ls"], [/* 15 vars */]) = 0
uname({sys="Linux", node="dinkel", ...}) = 0
brk(0)                                  = 0x8058a88
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT ↩
↪(No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
…
write(1, "bin   dos   lib\t\tnano-hdb1.tar.g"..., 56) = 56
write(1, "boot  etc   lost+found\topt\t\t  sb"..., 52) = 52
write(1, "dev   home  mnt\t\tproc\t\t  SKELETO"..., 52) = 52
munmap(0x40012000, 4096)                = 0
exit_group(0)                           = ?
```

Nell'estratto mostrato si vede solo l'inizio e la fine del file. In particolare, all'inizio si riconosce l'utilizzo di file all'interno della directory '`/etc/`'; nella parte mancante si potrebbero notare anche i riferimenti alle librerie; infine, si vede il risultato emesso dal programma, costituito dall'elenco di file e directory, quindi la conclusione del programma stesso.

Strace può essere utile anche per chi non ha grande esperienza, per esempio per sapere dove un certo programma va a cercare certi file, o comunque per scoprire cosa c'è che impedisce il funzionamento di un programma.

Strace può essere usato anche per analizzare il funzionamento di un processo già attivo, con l'aiuto dell'opzione '**`-p`**':

```
$ strace -p 12345[Invio]
```

In questo caso, si vuole analizzare il funzionamento del processo elaborativo che ha il numero PID 12345. Con l'opzione '**`-e trace=read`**' si può limitare l'attenzione alla lettura dei dati dai vari descrittori:

```
$ strace -p 12345 -e trace=read[Invio]
```

In questo modo, si può vedere tutto ciò che viene letto dai vari descrittori del processo elaborativo numero 12345.

Strace può anche seguire i processi figli di un certo processo, utilizzando le opzioni '**-f**' e '**-F**':

```
$ strace -f -F -p 12345[Invio]
```

Con questo esempio, viene seguito il processo elaborativo numero 12345 e quelli che lui stesso avvia.

Per poter leggere ciò che fa un altro processo elaborativo, Strace deve essere avviato con i privilegi necessari. Per esempio, se Strace funziona con i privilegi dell'utente Tizio, può leggere i processi che sono stati avviati dallo stesso utente (a meno che sia l'utente '**root**' che avvia qualcosa con privilegi di un utente comune). In alcuni sistemi, Strace viene installato con il bit SUID attivato e la proprietà all'utente '**root**' (SUID-root). In questo modo, se tutti gli utenti possono avviare il programma, **chiunque può leggere ciò che fanno gli altri**, anche quando si inseriscono dati riservati come una parola d'ordine.

Tabella 10.57. Alcune opzioni per l'uso di Strace.

Opzione	Descrizione

`-f -F`
 	
`-fF`
	Segue anche i processi figli.

`-p` ***pid***	Segue il processo elaborativo indicato attraverso il suo numero PID.

Opzione	Descrizione
`-o` ***file***	Salva il risultato dell'analisi nel file indicato.
`-e trace=`***chiamata***	Tiene sotto controllo specificatamente la chiamata di sistema indicata.
`-e trace=open,close,read,write`	Tiene sotto controllo le chiamate di sistema principali per l'accesso ai file.
`-e trace=file`	Tiene sotto controllo tutte le chiamate di sistema che hanno per argomento il nome di un file.

Come esempio pratico in cui diventa molto importante l'uso di Strace si può considerare un programma che offre un servizio di rete, sotto il controllo di Inetd, il quale non funziona perché non trova un file (quindi si presume che fallisca la funzione ***open()***), ma non si sa di quale file si tratti. In questo caso, non è possibile conoscere il numero PID del processo elaborativo, perché viene avviato di volta in volta da Inetd, pertanto occorre avvalersi della coppia di opzioni '**`-f`**' e '**`-F`**':

```
$ strace -f -F -e open -p $(pidof inetd) [Invio]
```

Per semplificare il lavoro, si lascia al programma '**`pidof`**' il compito di determinare il numero PID del processo corrispondente a Inetd.

10.7 Invio di segnali ai processi

I segnali sono dei numeri ai quali i programmi attribuiscono significati determinati, relativi a quanto accade nel sistema. I segnali rappresentano sia un'informazione che un ordine: nella maggior parte dei casi i programmi possono intercettare i segnali e compiere delle operazioni correlate prima di adeguarsi al nuovo stato, oppure addirittura rifiutare gli ordini; in altri casi sono sottomessi immediatamente agli ordini. La tabella 10.3, apparsa all'inizio del capitolo, riassume i segnali descritti dallo standard POSIX, mentre l'elenco completo può essere ottenuto consultando la pagina di manuale *signal(7)*.

I numeri dei segnali sono stati abbinati a nomi standard che ne rappresentano in breve il significato (in forma di abbreviazione o di acronimo). I numeri dei segnali non sono standard tra i vari sistemi Unix e dipendono dal tipo di architettura hardware utilizzata. Anche all'interno di GNU/Linux stesso ci possono essere differenze a seconda del tipo di macchina che si utilizza. Questo particolare è importante sia per giustificare il motivo per cui è opportuno fare riferimento ai segnali in forma verbale, sia per ricordare la necessità di fare attenzione con i programmi che richiedono l'indicazione di segnali esclusivamente in forma numerica (per esempio '**`top`**').

10.7.1 Segnali attraverso la tastiera

Alcuni segnali possono essere inviati al programma con il quale si interagisce attraverso delle combinazioni di tasti. Di solito si invia un segnale '**`SIGINT`**' attraverso la combinazione [*Ctrl c*], un segnale '**`SIGTSTP`**' attraverso la combinazione [*Ctrl z*] e un segnale '**`SIGQUIT`**' attraverso la combinazione [*Ctrl* \].

L'effetto di queste combinazioni di tasti dipende dalla configurazione della linea di terminale. Questa può essere controllata o modificata attraverso il programma '**stty**' (14.7.2). Come si può vedere dall'esempio seguente, alcune combinazioni di tasti (rappresentate nella forma **^*x***) sono associate a delle funzioni. Nel caso di quelle appena descritte, le funzioni sono '**intr**', '**susp**' e '**quit**'.

```
$ stty -a [Invio]

  speed 38400 baud; rows 25; columns 80; line = 0;
  intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
  eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S;
  susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O;
  min = 1; time = 0;
  -parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
  -ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr
  icrnl ixon ixoff -iuclc -ixany -imaxbel
  opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0
  cr0 tab0 bs0 vt0 ff0 isig icanon -iexten echo echoe echok
  -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Riquadro 10.59. Le origini dei segnali dalla tastiera.

L'idea del segnale inviato attraverso la tastiera deriva dall'uso della telescrivente, con la quale, alcuni codici servivano per ottenere delle funzioni, senza generare un carattere da stampare. Questi codici fanno parte dello standard ASCII e, oltre ad avere una corrispondenza numerica, hanno un nome e una rappresentazione nella forma <^*x*>. Per esempio, il codice corrispondente al numero 3_{16}, si indica con il nome <*ETX*> (*End of text*) e viene rappresentato come <^*c*>.
Quando dalla telescrivente si passa alla tastiera di un elaboratore che utilizza un sistema operativo Unix, occorre considerare che i tasti e le combinazioni di tasti sono configurabili, pertanto vanno associati a delle funzioni della tastiera. Pertanto, alcune funzioni che non corrispondono alla produzione di codici che generano un carattere tipografico, vengono associate con '**`stty`**' all'invio di un segnale al processo in funzione, ovvero al processo che usa la tastiera stessa. Quindi, la combinazione di tasti [*Ctrl c*] si presume sia associata alla funzione <*Control_c*> che '**`stty`**' indica come '**`^C`**' seguendo la tradizione della telescrivente.

10.7.2 Segnali attraverso la shell

Le shell offrono generalmente dei comandi interni per l'invio di segnali ai processi da loro avviati. In particolare, quelle che come Bash sono in grado di gestire i *job*, ovvero i gruppi di elaborazione, utilizzano i segnali in modo trasparente per fare riprendere un processo sospeso.

Per esempio, nel caso di Bash, se un processo viene sospeso attraverso la combinazione [*Ctrl z*], cosa che dovrebbe generare un segnale '**`SIGTSTP`**' (in base alla configurazione della linea di terminale), questo può essere riportato in primo piano e in funzione, attraverso il comando '**`fg`**', con il quale in pratica si invia al processo un segnale

‘**SIGCONT**’.

10.7.3 Comandi «kill...»

«

Il modo normale per inviare un segnale a un processo è l’uso di ‘**kill**’: questo, a seconda dei casi, può essere un comando interno di shell o un programma. Il nome ‘**kill**’ deriva in particolare dall’effetto che si ottiene utilizzandolo senza l’indicazione esplicita di un segnale da inviare: quello predefinito è ‘**SIGTERM**’ attraverso il quale si ottiene normalmente la conclusione del processo destinatario.

Attraverso ‘**kill**’ si riesce solitamente a ottenere un elenco dei segnali disponibili con il loro numero corrispondente. Ciò è molto importante per conoscere esattamente quale numero utilizzare con i programmi che non permettono l’indicazione dei segnali in forma verbale.

```
$ kill -l [Invio]

 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
```

In un sistema GNU/Linux sono disponibili anche altri programmi con funzioni analoghe: ‘**killall**’ e ‘**killall5**’. Per cominciare, quella che segue è la sintassi del comando ‘**kill**’:

```
kill [opzioni] [numero_pid...]
```

Se non viene specificato, il segnale predefinito è '**SIGTERM**' che normalmente procura la conclusione dell'esecuzione dei processi destinatari. Questo giustifica il nome '**kill**'.[13]

Tabella 10.61. Alcune opzioni.

Opzione	Descrizione
`-s` ***segnale***	Specifica il nome o il numero del segnale da inviare.
`-l`	Mostra l'elenco dei segnali disponibili con i numeri corrispondenti.

Segue la descrizione di alcuni esempi.

- `$` **`kill -s SIGHUP 1203`** [*Invio*]

  Invia il segnale '**SIGHUP**' al processo corrispondente al numero 1203.

- `$` **`kill -s 1 1203`** [*Invio*]

  Esattamente come nell'esempio precedente, in un sistema GNU/Linux.

- `$` **`kill -l`** [*Invio*]

  Mostra l'elenco dei segnali disponibili.

Negli esempi di questo documento viene indicato spesso il segnale da inviare senza l'opzione '**-s**', usando piuttosto la forma '***-segnale***'. Questo modo di indicare il segnale riguarda il comando interno omonimo della shell Bash.

Il programma '**killall**' invia un segnale a tutti i processi che eseguono i comandi specificati.

```
killall [opzioni] [-segnale] [comando...]
```

Si utilizza quindi '**killall**' per inviare un segnale a dei processi identificati per nome. Se non viene specificato il segnale da inviare, si utilizza '**SIGTERM**'. I segnali possono essere indicati per nome o per numero.[14]

Tabella 10.62. Alcune opzioni.

Opzione	Descrizione
`-s` ***segnale***	Specifica il nome o il numero del segnale da inviare.
`-l`	Mostra l'elenco dei segnali disponibili con i numeri corrispondenti.

L'esempio seguente invia il segnale '**SIGHUP**' a tutti i processi avviati con il comando '**pippo**'. I processi soggetti a questo sono solo quelli che appartengono all'utente che invia il segnale.

```
$ killall -HUP pippo [Invio]
```

Il programma '**killall5**'[15] consente di inviare un segnale a tutti i processi elaborativi in funzione, esclusi quelli della propria sessione e quelli strettamente connessi con il kernel (*kernel thread*). Potrebbe essere utilizzato nella procedura di arresto del sistema.

```
killall5 -n_segnale
```

È il caso di rammentare che altri programmi potrebbero avere la facoltà di inviare segnali ai processi, come funzione accessoria. Per esempio '**fuser**', usato con l'opzione '**-k**' può inviare un segnale ai

processi che utilizzano un certo file (sezione 10.4.1).

10.8 Controllo dei «job» di shell

Attraverso alcune shell è possibile gestire i *job*, ovvero i «gruppi di elaborazione», che in questo caso rappresentano raggruppamenti di processi generati da un solo comando impartito alla shell stessa.

La shell Bash e in generale le shell POSIX, oltre alla shell Korn e alla shell C, gestiscono questo tipo di raggruppamenti di processi. Nelle sezioni seguenti si fa riferimento al comportamento di Bash (in qualità di shell POSIX), ma la maggior parte di quanto spiegato in queste sezioni vale anche per le shell Korn e C (‘**ksh**’ e ‘**csh**’).

Non si deve confondere il gruppo di elaborazione di una shell con un processo. Un processo è un singolo eseguibile messo in funzione: se questo a sua volta avvia un altro eseguibile, viene generato un nuovo processo a esso associato. Un gruppo di elaborazione rappresenta tutti i processi che vengono generati da un comando impartito tramite la shell. Basta immaginare cosa succede quando si utilizza un condotto di programmi (*pipeline*), dove l’output di un programma è l’input del successivo.

10.8.1 Processi in primo piano e processi sullo sfondo

L’attività di un gruppo di elaborazione può avvenire in primo piano (*foreground*) o sullo sfondo (*background*). Nel primo caso, il gruppo di elaborazione impegna la shell e quindi anche il terminale, mentre nel secondo la shell è libera da impegni e così anche il

terminale. Di conseguenza, non ha senso pretendere da un programma che richiede l'interazione continua con l'utente che possa anche funzionare sullo sfondo.

Se un programma richiede dati dallo standard input o ha la necessità di emettere dati attraverso lo standard output o lo standard error, per poterlo avviare come gruppo di elaborazione sullo sfondo, bisogna almeno provvedere a ridirigere l'input e l'output.

10.8.2 Avvio di un gruppo di elaborazione sullo sfondo

«

Un programma è avviato esplicitamente come gruppo di elaborazione sullo sfondo quando alla fine della riga di comando viene aggiunto il simbolo '**&**'. Per esempio:

```
# make bzImage > ~/make.msg & [Invio]
```

avvia sullo sfondo il comando '**make bzImage**', per generare un kernel, dirigendo lo standard output verso un file per consentire un controllo successivo dell'esito della compilazione.

Dopo l'avvio di un programma come gruppo di elaborazione sullo sfondo, la shell restituisce una riga contenente il numero del gruppo di elaborazione e il numero del processo terminale generato da questo (il numero PID). L'esempio seguente rappresenta il gruppo di elaborazione numero uno che termina con il processo 173:

```
[1] 173
```

Se si avvia un gruppo di elaborazione sullo sfondo, quando a un certo punto questo ha la necessità di emettere dati attraverso lo standard output o lo standard error e tali flussi di dati non sono stati ridiretti, si ottiene una segnalazione simile a quella seguente:

```
[1]+ Stopped (tty output) pippo
```

Nell'esempio, il gruppo di elaborazione avviato con il comando '**pippo**' si è bloccato in attesa di poter emettere dell'output. Nello stesso modo, se viene avviato un gruppo di elaborazione sullo sfondo che a un certo punto ha la necessità di ricevere dati dallo standard input, ma questo non è stato ridiretto, si ottiene una segnalazione simile alla seguente:

```
[1]+ Stopped (tty input) pippo
```

10.8.3 Sospensione di un gruppo di elaborazione in primo piano

Se è stato avviato un gruppo di elaborazione in primo piano e si desidera sospenderne l'esecuzione, si può inviare attraverso la tastiera il carattere '**susp**', che di solito si ottiene con la combinazione [*Ctrl z*]. Il gruppo di elaborazione viene sospeso e posto sullo sfondo. Quando un gruppo di elaborazione viene sospeso, la shell genera una riga come nell'esempio seguente dove il gruppo di elaborazione '**pippo**' è stato sospeso:

```
[1]+ Stopped pippo
```

10.8.4 Utilizzo di «jobs»

Il comando di shell '**jobs**', permette di conoscere l'elenco dei gruppi di elaborazione esistenti e il loro stato.

`jobs` [*opzioni*] [*job*]

Per poter utilizzare il comando '**jobs**' occorre che non ci siano altri gruppi di elaborazione in esecuzione in primo piano, di conseguenza, quello che si ottiene è solo l'elenco di quelli sullo sfondo.

Tabella 10.67. Alcune opzioni.

Opzione	Descrizione
`-l`	Permette di conoscere anche i numeri PID dei processi di ogni gruppo di elaborazione.
`-p`	Emette solo il numero PID del processo iniziale di ogni gruppo di elaborazione.

Segue la descrizione di alcuni esempi.

- `$` **`jobs`** [*Invio*]

  Si ottiene l'elenco normale dei gruppi di elaborazione sullo sfondo. Nel caso dell'esempio seguente, il primo gruppo di elaborazione è in esecuzione, il secondo è sospeso in attesa di poter emettere l'output, l'ultimo è sospeso in attesa di poter ricevere l'input.

```
[1]   Running                 yes >/dev/null &
[2]-  Stopped (tty output)    mc
[3]+  Stopped (tty input)     unix2dos
```

  Per comprendere l'utilizzo dell'opzione '**`-l`**' e dell'opzione '**`-p`**', occorre avviare sullo sfondo qualche comando un po' articolato.

- `$` **`yes | cat | sort > /dev/null &`** [*Invio*]

```
[1] 594
```

- `$` **`yes | cat > /dev/null &`** [*Invio*]

```
[2] 596
```

- `$` **`jobs -l`** [*Invio*]

```
[1]-   592 Running                 yes
       593                       | cat
       594                       | sort >/dev/null &
[2]+   595 Running                 yes
       596                       | cat >/dev/null &
```

  Come si può osservare, l'opzione '**`-l`**' permette di avere informazioni più dettagliate su tutti i processi che appartengono ai vari gruppi di elaborazione presenti.

- `$` **`jobs -p`** [*Invio*]

  Si ottiene soltanto l'elenco dei numeri PID del processo iniziale di ogni gruppo di elaborazione.

```
592
595
```

10.8.5 Riferimenti ai gruppi di elaborazione

L'elenco di gruppi di elaborazione ottenuto attraverso il comando '**`jobs`**', mostra in particolare il simbolo '**`+`**' a fianco del numero del gruppo di elaborazione attuale ed eventualmente il simbolo '**`-`**' a fianco di quello che diventerebbe il gruppo di elaborazione attuale se il primo termina o viene comunque eliminato.

Il gruppo di elaborazione attuale è quello a cui si fa riferimento in modo predefinito tutte le volte che un comando richiede l'indicazione di un gruppo di elaborazione, ma questo non viene fornito.

Di norma si indica un gruppo di elaborazione con il suo numero preceduto dal simbolo '`%`', ma si possono anche utilizzare altri metodi elencati nella tabella 10.73.

Tabella 10.73. Elenco dei parametri utilizzabili come riferimento ai gruppi di elaborazione della shell.

Simbolo	Descrizione
`%`***n***	Il gruppo di elaborazione con il numero indicato dalla lettera ***n***.
`%`***stringa***	Il gruppo di elaborazione il cui comando inizia con la stringa indicata.
`%?`***stringa***	Il gruppo di elaborazione il cui comando contiene la stringa indicata.
`%%`	Il gruppo di elaborazione attuale.
`%+`	Il gruppo di elaborazione attuale.
`%-`	Il gruppo di elaborazione precedente a quello attuale.

10.8.6 Comando «fg»

Il comando interno '**`fg`**' porta in primo piano un gruppo di elaborazione che in precedenza è stato messo sullo sfondo. Se non viene specificato il gruppo di elaborazione su cui agire, si intende quello attuale.

`fg` [***job***]

10.8.7 Comando «bg»

Il comando interno '**bg**' permette di fare riprendere (sullo sfondo) l'esecuzione di un gruppo di elaborazione sospeso. Ciò è possibile solo se il gruppo di elaborazione in questione non è in attesa di un input o di poter emettere l'output. Se non si specifica il gruppo di elaborazione, si intende quello attuale.

```
bg [job]
```

Quando si utilizza la combinazione [*Ctrl z*] per sospendere l'esecuzione di un gruppo di elaborazione, questo viene messo sullo sfondo e diviene il gruppo di elaborazione attuale. Di conseguenza, è normale utilizzare il comando '**bg**' subito dopo, senza argomenti, in modo da fare riprendere il gruppo di elaborazione appena sospeso.

10.8.8 Comando «kill»

Il comando interno '**kill**' funziona quasi nello stesso modo del programma omonimo. Di solito, non ci si rende conto che si utilizza il comando e non il programma. Il comando '**kill**' in particolare, rispetto al programma, permette di inviare un segnale ai processi di un gruppo di elaborazione, indicando direttamente il gruppo di elaborazione stesso.

```
kill [-s segnale | -segnale] [job]
```

Quando si vuole eliminare tutto un gruppo di elaborazione, a volte non è sufficiente un segnale '**SIGTERM**'. Se necessario si può utilizzare il segnale '**SIGKILL**' (con prudenza però).

Segue la descrizione di alcuni esempi.

- `$` **`kill -KILL %1`** [*Invio*]

  Elimina i processi abbinati al gruppo di elaborazione numero uno, inviando il segnale '**SIGKILL**'.

- `$` **`kill -9 %1`** [*Invio*]

  Elimina i processi abbinati al gruppo di elaborazione numero uno, inviando il segnale '**SIGKILL**', espresso in forma numerica.

10.9 Cattura dei segnali con la shell

«

Attraverso il comando interno '**trap**' di una shell standard, è possibile catturare ed eventualmente attribuire un comando (comando interno, funzione o programma) a un segnale particolare. In questo modo uno script può gestire i segnali. L'esempio seguente ne mostra uno ('**trappola**') in grado di reagire ai segnali '**SIGUSR1**' e '**SIGUSR2**' emettendo semplicemente un messaggio.

```
#!/bin/sh
trap 'echo "Ho catturato il segnale SIGUSR1"' SIGUSR1
trap 'echo "Ho catturato il segnale SIGUSR2"' SIGUSR2
# Ripete continuamente.
while [ 0 ]
do
    # Esegue un'operazione inutile.
    NULLA="ciao"
done
```

Supponendo di avere avviato lo script nel modo seguente, ottenendo il numero PID 1234, si osservi cosa accade:

`$` **`trappola &`** [*Invio*]

```
$ kill -s SIGUSR1 1234 [Invio]

  Ho catturato il segnale SIGUSR1
$ kill -s SIGUSR2 1234 [Invio]

  Ho catturato il segnale SIGUSR2
```

10.9.1 Comando «trap»

Il comando espresso come argomento di '**trap**' viene eseguito quando la shell riceve il segnale o i segnali indicati.

```
trap [-l] [comando] [segnale]
```

Se non viene fornito il comando, o se al suo posto si mette un trattino ('**-**'), tutti i segnali specificati sono riportati al loro valore originale (i valori che avevano al momento dell'ingresso nella shell), cioè riprendono il loro significato normale. Se il comando fornito corrisponde a una stringa nulla, il segnale relativo viene ignorato dalla shell e dai comandi che questo avvia. Il segnale può essere espresso in forma verbale (per nome) o con il suo numero. Se il segnale è '**EXIT**', pari a zero, il comando viene eseguito all'uscita della shell.

Se viene utilizzato senza argomenti, '**trap**' emette la lista di comandi associati con ciascun numero di segnale.

Segue la descrizione di alcuni esempi.

- `$ trap 'ls -l' SIGUSR1 [Invio]`

  Se la shell riceve un segnale '**SIGUSR1**' esegue '**ls -l**'.

- `$ trap '' SIGUSR1 [Invio]`

La shell e tutti i processi figli ignorano il segnale '**SIGUSR1**'.

10.10 Trasformare dei programmi comuni in demoni

«

Per far sì che un programma, fatto per funzionare normalmente in modo interattivo, svolga il suo lavoro sullo sfondo, non è sempre sufficiente avviarlo dalla shell con l'operatore '**&**', dopo avere ridiretto opportunamente i suoi flussi standard, perché rimane la dipendenza dal processo che lo genera; infatti, al termine del funzionamento della shell il processo sullo sfondo sarebbe coinvolto nella stessa sorte. Per ovviare a tale inconveniente, è necessario che i processi diventino figli di '**init**'.

10.10.1 Utilizzo di «nohup»

«

Il programma '**nohup**' esegue un comando facendo in modo che questo non sia interessato dai segnali di interruzione di linea ('**SIGHUP**'). In questo senso, '**nohup**' permette di avviare dei processi che non devono interrompersi nel momento in cui l'utente che li avvia termina la sua sessione di lavoro (chiude la connessione con il terminale). Naturalmente, questo ha senso se i programmi vengono avviati sullo sfondo.[16]

```
nohup comando [argomenti]
```

In base a questo principio, cioè quello per cui si usa '**nohup**' per avviare un programma sullo sfondo in modo che continui a funzionare anche quando l'utente si scollega, la priorità di esecuzione viene modificata, aumentando il valore nice di cinque unità.

Il comando indicato come argomento non viene messo automaticamente sullo sfondo, per ottenere questo occorre aggiungere il simbolo '**&**' (e-commerciale) alla fine della riga di comando. Quando il comando indicato come argomento utilizza il terminale per emettere l'output, sia lo standard output, sia lo standard error vengono ridiretti verso il file '`./nohup.out`', oppure, se i permessi non lo consentono, verso il file '`~/nohup.out`'. Se questo file esiste già i dati vengono aggiunti.

Segue un esempio che mostra come si comporta '**nohup**'. Si comincia dall'avvio di una nuova copia della shell Bash nel modo seguente:

```
$ bash [Invio]
```

Viene avviato sullo sfondo il programma '**yes**' e il suo output viene semplicemente ridiretto verso '`/dev/null`':

```
$ nohup yes > /dev/null & [Invio]

  [1] 1304
```

Il processo corrispondente ha il numero PID 1304. Si controlla lo stato dei processi attraverso '**ps**':

```
$ ps [Invio]

    PID TTY STAT  TIME COMMAND
  …
   1304   1 R N  1:55 yes
  …
```

Dalla colonna '**STAT**' si può osservare che '**yes**' ha un valore nice positivo (si osserva per questo la lettera '**N**'). Si controlla lo stato dei processi attraverso '**pstree**':

```
$ pstree -p[Invio]

  init(1)-+-…
          |
          …
          |-login(370)---bash(387)---bash(1303)-+-pstree(1341)
          |                                     `-yes(1304)
          …
```

Si può osservare che '**yes**' è un processo figlio della shell Bash (l'eseguibile '**bash**') avviata poco prima. Si conclude l'attività della shell provocando un segnale di interruzione di linea per i processi che dipendono da questa:

```
$ exit[Invio]
```

Si controlla nuovamente lo stato dei processi attraverso '**pstree**':

```
$ pstree -p[Invio]

  init(1)-+-…
          |
          …
          |-login(370)---bash(387)---pstree(1359)
          …
          |-yes(1304)
          …
```

Al termine, '**yes**' risulta essere un processo figlio del processo principale (Init, ovvero l'eseguibile '**init**').

Probabilmente, facendo qualche esperimento, si può osservare che i processi sullo sfondo non terminano la loro esecuzione quando si conclude la sessione di lavoro della shell che li ha avviati, senza bisogno di utilizzare '**nohup**'. Tuttavia ci sono situazioni in cui '**nohup**' è indispensabile. Per esempio, se si sta lavorando con l'ambiente grafico X e si chiude una finestra di terminale, un eventuale programma sullo sfondo viene eliminato sicuramente, a meno di usare '**nohup**'.

10.10.2 Utilizzo di «daemon»

Il programma '**daemon**' esegue un comando, trasformando se stesso e ciò che avvia nell'equivalente di un demone vero e proprio. Il lavoro di '**daemon**' è molto più sofisticato rispetto a '**nohup**' e per l'avvio del comando non serve ridirigere esplicitamente i flussi di dati standard e nemmeno cercare di mettere il processo sullo sfondo, perché a questo provvede direttamente '**daemon**'.

```
daemon [opzioni] [[--] comando_e_argomenti]
```

A titolo di esempio viene mostrata una situazione tipica di utilizzo di '**daemon**', in cui si vuole avviare il programma '**lambdamoo**'. Il programma '**lambdamoo**' offre un servizio attraverso la rete, ma se lo si avvia rimane in primo piano e non sarebbe sufficiente l'uso di '**nohup**' per sganciarlo dal terminale:

```
# daemon -u lambdamoouser -- ↩
↪        lambdamoo -l /var/log/lambdamoo.log ↩
↪        /var/run/lambdamoo/enCorumbe26012008.prv ↩
↪        /var/run/lambdamoo/enCorumbe26022008.prv [Invio]
```

In questo modo, il programma '**lambdamoo**' viene anche avviato con i privilegi di un utente specifico ('**lambdamoouser**') e il resto riguarda l'uso del programma stesso.

Dopo l'avvio di questo comando di esempio, si può osservare lo stato dei processi con '**pstree**':

```
init-+-…
     |
     …
     |-daemon---lambdamoo
     …
     |-2*[lambdamoo]
     …
```

In questo caso, di fatto appaiano altri due processi di '**lambdamoo**', figli di '**init**', dipende da '**lambdamoo**' stesso, in quanto il compito di '**daemon**' si limita all'avvio di quel processo che diviene suo figlio.

Tabella 10.82. Alcune opzioni per l'uso di '**daemon**'.

Opzione	Descrizione
`-n` ***nome*** `--n=`***nome***	Attribuisce un nome al lavoro, da usare per la creazione di un file in '`/var/run/`' (un file lucchetto, ovvero *lock file*) con il numero del processo avviato, allo scopo di evitare l'avvio di altri processi simultanei con quello stesso nome.
`-u` ***utente*** [***.gruppo***] `--user=`***utente*** [***.gruppo***]	Avvia il comando con i privilegi dell'utente indicato ed eventualmente anche del gruppo.

Opzione	Descrizione
`-R` ***dir*** `--chroot=`***dir***	Avvia il comando facendo il modo che la directory indicata diventi per lui quella principale (la radice).
`-D` ***dir*** `--chdir=`***dir***	Avvia il comando facendo il modo che la directory indicata diventi per lui quella corrente, di funzionamento.
`-m` ***umask*** `--umask=`***umask***	Avvia il comando facendo il modo che la maschera dei permessi diventi quella indicata.
`-r` `--respawn`	Riavvia il comando se questo termina di funzionare per qualche ragione.
`-o` ***file*** `--output=`***file***	Avvia il comando, facendo in modo che quanto prodotto da questo, attraverso lo standard output e lo standard error, venga diretto al file specificato.
`-o` ***servizio . priorità*** `--output=`***servizio . priorità***	Avvia il comando, facendo in modo che quanto prodotto da questo, attraverso lo standard output e lo standard error, venga diretto al registro del sistema (*syslog*), con il servizio e la priorità specificati (*facility* e *priority*).
`-O` ***file*** `--stdout=`***file***	Avvia il comando, facendo in modo che quanto prodotto da questo attraverso lo standard output, venga diretto al file specificato.

Opzione	Descrizione
`-O` ***servizio . priorità*** `--stdout=`***servizio . priorità***	Avvia il comando, facendo in modo che quanto prodotto da questo attraverso lo standard output, venga diretto al registro del sistema (*syslog*), con il servizio e la priorità specificati (*facility* e *priority*).
`-E` ***file*** `--stderr=`***file***	Avvia il comando, facendo in modo che quanto prodotto da questo attraverso lo standard error, venga diretto al file specificato.
`-E` ***servizio . priorità*** `--stderr=`***servizio . priorità***	Avvia il comando, facendo in modo che quanto prodotto da questo attraverso lo standard error, venga diretto al registro del sistema (*syslog*), con il servizio e la priorità specificati (*facility* e *priority*).

[1] Il concetto di priorità fa riferimento a una sequenza ordinata di elementi: il primo, cioè quello che ha precedenza sugli altri, è quello che ha il valore inferiore.

[2] A questo proposito, la distribuzione SuSE colloca questi script nella directory '`/sbin/init.d/`'.

[3] **Procps ps** GNU LGPL

[4] **Psmisc** software libero con licenza speciale

[5] **Procps top** GNU GPL

[6] **Htop** GNU GPL

[7] **System V Init** GNU GPL

[8] **Psmisc** GNU GPL

[9] **Lsof** software libero con licenza speciale

[10] **Procps uptime** GNU GPL

[11] **Procps free** GNU GPL

[12] **Strace** software libero con licenza speciale

[13] **Procps kill** GNU GPL

[14] **Psmisc** software libero con licenza speciale

[15] **System V Init** GNU GPL

[16] **GNU core utilities** GNU GPL