

Controllo di processi

Ogni processo del sistema ha un **PID** (**P**rocess **I**dentify **N**umber).

Ogni processo può generare nuovi processi (figli).

La radice della gerarchia di processi è il processo **init** con PID=1.

init è il primo processo che parte al boot di sistema.

Il comando `ps` fornisce i processi presenti nel sistema:

```
user> ps    # fornisce i processi dell'utente associati al terminale corrente
```

PID	TTY	TIME	CMD
23228	pts/15	0:00	xdvi.bin
9796	pts/15	0:01	bash
23216	pts/15	0:04	xemacs-2
9547	pts/15	0:00	csch

Legenda: PID = PID; TTY = terminale (virtuale); TIME = tempo di CPU utilizzato; CMD = comando che ha generato il processo.

Per ottenere il nome del terminale corrente:

```
user> tty
/dev/pts/15
```

Il comando `ps` e sue varianti, I

Per ottenere tutti i processi nel sistema associati ad un terminale (`-a`), full listing (`-f`):

```
user> ps -af
```

	UID	PID	PPID	C	STIME	TTY	TIME	CMD
	lenisa	10922	9560	0	Oct 17	pts/17	0:00	bash
	pietro	23410	23409	0	11:07:08	pts/26	0:01	xdvi.bin -name xdvi main.dvi
	root	24188	9807	0	12:34:10	pts/13	0:00	ps -af
							

Legenda: UID = User Identifier; PPID = Parent PID; c = informazione obsoleta sullo scheduling; STIME = data di inizio del processo.

Il comando ps e sue varianti, II

Per ottenere tutti i processi nel sistema, anche non associati ad un terminale (-e), long listing (-l):

```
user> ps -el
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
19	T	0	0	0	0	0	SY	?	0		?	0:12	sched
8	S	0	1	0	0	41	20	?	100	?	?	0:03	init
8	S	140	12999	12997	0	56	20	?	278	?	pts/12	0:00	tcsh
8	R	159	9563	9446	0	50	20	?	2110		?	0:30	acroread
. . . .													

Legenda: F = flag obsoleti; S = stato del processo (T=stopped); PRI = priorità; NI = nice value (usato per modificare la priorità); ADDR = indirizzo in memoria; SZ = memoria virtuale usata; WCHAN = evento su cui il processo è sleeping.

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza Ctrl-k dal terminale stesso su cui il processo è in esecuzione;
- il comando `kill` seguito dal PID del processo (da qualsiasi terminale):

```
user> ps
```

PID	TTY	TIME	CMD
.....			
28015	pts/14	0:01	xemacs
.....			

```
user> kill 28015
```

- il comando `kill` con il segnale `SIGKILL`

```
user> kill -9 28015
```

```
user> kill -s kill 28015
```

Processi in background

Un comando (pipeline, sequenza) seguito da & dà luogo ad uno o più **processi in background**. I processi in background sono eseguiti in una **sottoshell**, in **parallelo** al processo padre (la shell) e **non** sono controllati da tastiera.

I processi in background sono quindi utili per eseguire task in parallelo che non richiedono controllo da tastiera.

```
user> xemacs &
```

```
[1] 24760
```

[1] è il numero del job, 24760 il PID del processo

```
user> xemacs &
```

```
user> ls -R / >tmp 2>err &
```

Il comando jobs mostra la lista dei job in esecuzione:

```
user> jobs
```

```
[1]      Running          xemacs &
```

```
[2]-    Running          xemacs &
```

```
[3]+    Running          ls -R / >tmp 2>err
```

Controllo di job

Un job si può **sospendere** e poi **rimandare in esecuzione**

```
user> cat >temp      # job in foreground
```

```
Ctrl-z    # sospende il job
```

```
[1]+ Stopped
```

```
user> jobs
```

```
[1]+ Stopped      cat >temp
```

```
user> fg      # fa il resume del job in foreground
```

```
Ctrl-z    # sospende il job
```

```
user> bg      # fa il resume del job in background
```

```
user> kill %1    # termina il job 1
```

```
[1]+ Terminated
```

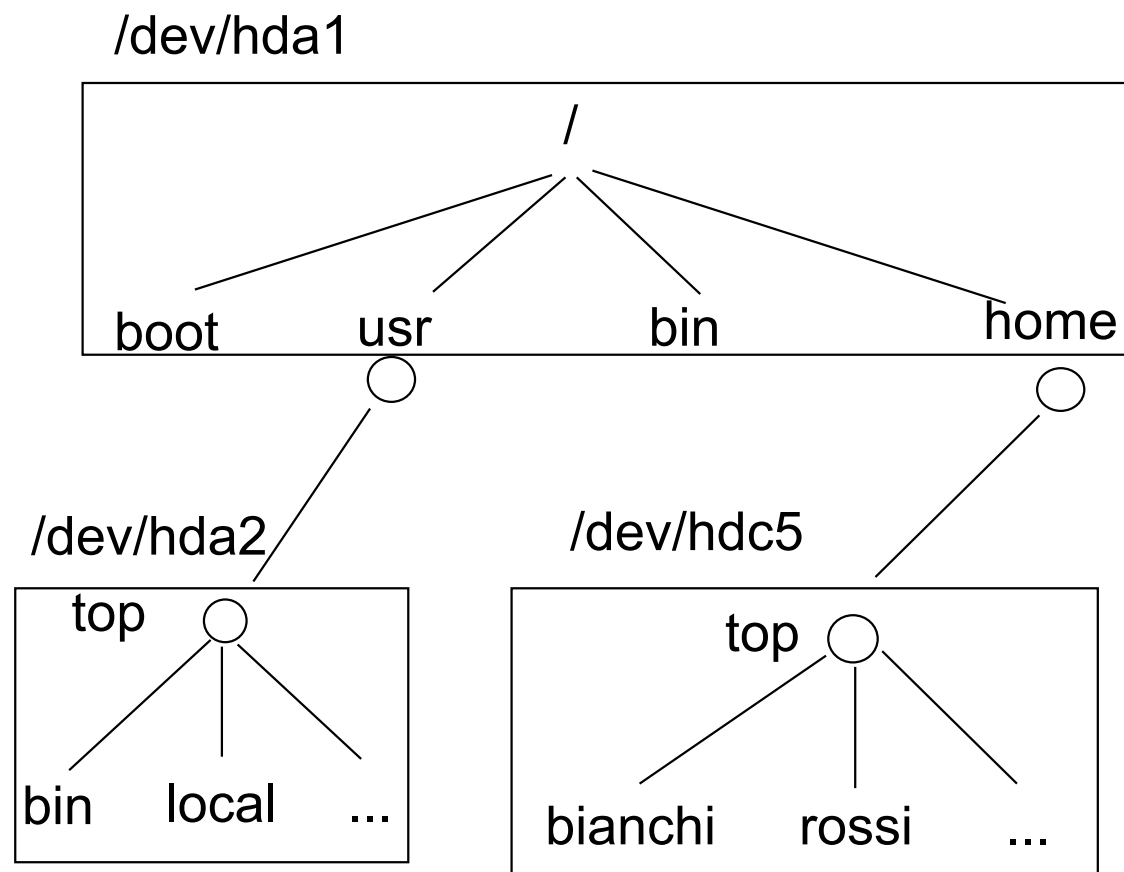
Il filesystem di Unix/Linux (I)

- Comunemente, in un elaboratore l'informazione è memorizzata in modo permanente nei **dischi fissi**.
- Ogni disco fisso può essere suddiviso in **partizioni**.
- Ogni partizione può contenere un filesystem con una propria **top level directory**.

Sorge quindi il problema di come permettere agli utenti di accedere ai vari filesystem contenuti nelle differenti partizioni:

- la prima possibilità consiste nell'avere **root directory distinte** (e.g. in Windows: C:\, D:\ ecc.): una per ogni partizione; quindi per riferirsi ad un file, bisogna usare un pathname che parta dalla root directory giusta (e.g. D:\Doc\bilancio.xls).
- Unix/Linux invece fa in modo che i diversi filesystem vengano combinati in un'**unica struttura gerarchica**, “montando” la top level directory di una partizione come foglia del filesystem di un'altra partizione.

Il filesystem di Unix/Linux (II)



dove `/dev/hda1`, `/dev/hda2` e `/dev/hdc5` sono partizioni differenti. Le informazioni su quali filesystem montare al boot ed in che modo sono contenute nel file `/etc/fstab`. Il comando per montare i filesystem è `mount <file speciale> <mount point>` e, solitamente, solo l'utente `root` può utilizzarlo. `mount` senza argomenti elenca i filesystem in uso nel sistema.

Controllo dello spazio su disco

Per controllare la quantità di spazio su disco in uso:

```
user> df
```

/	(/dev/dsk/c0t0d0s0):	2231020 blocks	304297 files
/proc	(/proc):	0 blocks	11692 files
/opt	(/dev/dsk/c0t0d0s3):	2486076 blocks	325130 files
/usr/local	(/dev/dsk/c0t0d0s7):	4067088 blocks	430227 files
/opt/solaris2	(apphost:/opt/solaris2):	1995968 blocks	286013 files

Legenda: il primo campo contiene il nome del file system; il secondo il device corrispondente (eventualmente virtuale); il terzo il numero di blocchi occupati; il quarto il numero di inode.

Per controllare la quantità di spazio su disco utilizzata da una directory (in blocchi):

```
user> du LABORATORIO_S0
```

8	LABORATORIO_S0/LABS0/CVS
16884	LABORATORIO_S0/LABS0
16	LABORATORIO_S0/scriptColonne
14	LABORATORIO_S0/linguaggio_c
17342	LABORATORIO_S0

Monitoraggio della memoria

Il comando `top` fornisce informazioni sulla memoria utilizzata dai processi, che vengono aggiornate ad intervalli di qualche secondo. I processi sono elencati secondo la quantità di tempo di CPU utilizzata.

```
user> top
```

```
load averages:  0.68,  0.39,  0.27                                14:34:55
```

```
245 processes: 235 sleeping, 9 zombie, 1 on cpu
```

```
CPU states: 91.9% idle,  5.8% user,  2.4% kernel,  0.0% iowait,  0.0% swap
```

```
Memory: 768M real, 17M free, 937M swap in use, 759M swap free
```

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
12887	root	1	59	0	65M	56M	sleep	105:00	3.71%	Xsun
4210	lenisa	1	48	0	2856K	2312K	cpu	0:00	1.50%	top
9241	root	1	59	0	35M	26M	sleep	15:58	1.47%	Xsun
24389	pietro	4	47	0	28M	25M	sleep	16:30	0.74%	opera
.....										

Legenda: la prima riga indica il carico del sistema nell'ultimo minuto, negli ultimi 5 minuti, negli ultimi 15 minuti, rispettivamente; il carico è espresso come numero di processori necessari per far girare tutti i processi a velocità massima; alla fine della prima riga c'è l'ora; la seconda contiene numero e stato dei processi nel sistema; la terza l'utilizzo della CPU; la quarta informazioni sulla memoria; le restanti righe contengono informazioni sui processi (THR=thread, RES=resident)

Esercizi

- Qual è la differenza tra **programma** e **processo**?
- Qual è la differenza tra **processo** e **job**?
- Scoprire quanto spazio occupa il contenuto della propria home directory. Esiste un modo per ottenere in output soltanto il numero di blocchi (evitando di visualizzare informazioni ulteriori)?
- Scrivere una pipeline che fornisca in output il numero di processi appartenenti all'utente `root`.
- Il comando

```
> emacs &
```

provoca l'avvio di un processo in **background**. Invece il comando

```
> emacs
```

provoca l'avvio di un processo in **foreground**. Come si può mandare tale processo in esecuzione in background in modo da rendere il terminale nuovamente disponibile per l'invio di ulteriori comandi?