

# Controllo dei processi

# I processi

- programmi in esecuzione
- lo stesso programma può corrispondere a diversi processi (es., tanti utenti che usano emacs)
- ogni processo ha
  - process identification number (**PID**)
  - parent process identification number (**PPID**)
- tranne il processo *init*, che ha PID=1 e nessun PPID

# Attributi dei processi

- A ogni processo sono associati due utenti
  - **Real user**: utente che ha lanciato il processo
  - **Effective user**: utente che determina i diritti del processo
- quando il processo apre un file, vale l'effective user
- di solito, i due utenti coincidono
- se invece un file eseguibile ha il bit “*set user ID*” impostato, il corrispondente processo avrà:
  - Real user: utente che ha lanciato il processo
  - Effective user: utente proprietario dell'eseguibile

# Attributi dei processi

- Ogni processo ha anche due **gruppi** associati
  - **real group** ed **effective group**
- Un programma con “set user ID” è *ping*

```
> ls -l /bin/ping  
-rwsr-xr-x 1 root root 30804 2006-10-16 19:32 /bin/ping
```

- ping ha bisogno di partire con permessi di amministratore
  - ma per sicurezza rilascia questi permessi quasi subito

# ps [*selezione*] [*formato*]

## Selezione:

- *niente* i processi lanciati dalla shell corrente
- -u pippo i processi dell'utente pippo
- -A (All) tutti i processi

## Formato:

- *niente* PID, terminale, ora di esecuzione, comando
- -f (full) anche UID, PPID, argomenti
- -F (Full) anche altro
- -o elenco\_campi visualizza i campi specificati

# **ps** [*selezione*] [*formato*]

- Esempi
  - ps -u \$USER
  - ps -u \$USER -F
  - ps -u \$USER -o pid,cmd
  - ps -A -F
  - ps -A | less
  - ps -A | grep bash
  - pids=\$(ps -A -o pid)

# Esercizi su **ps**

- 1) Elencare tutte le istanze di bash in esecuzione
- 2) Elencare tutti gli utenti che hanno almeno un processo in esecuzione (senza ripetizioni)
- 3) Elencare i processi che non sono collegati a un terminale
- 4) Elencare i processi i cui eseguibili risiedono nella directory /sbin

# Controllo dei processi

- Normalmente, la shell aspetta che ogni comando termini (comando in *foreground*)
- Con “**comando&**”, la shell non aspetta (comando in *background*)
  - Il comando può comunque scrivere su standard output
- Esempio:

```
> ls -R /usr
...
> ls -R /usr > elenco &
[1] 11700
```

elenca ricorsivamente il contenuto di /usr e di tutte le sottodirectory

job  
number

PID

# Controllo dei processi

- **jobs** mostra i processi attualmente in background
- **fg** riporta in **foreground** l'ultimo processo lanciato in background
- **Ctrl-Z** **sospende** il processo attualmente in foreground
- **bg** manda in **background** l'ultimo processo sospeso
- **Ctrl-C** **termina** il processo attualmente in foreground

# Riferimenti

- Capitolo 4 di [Introduction to Linux]