LABORATORIO DI SISTEMI OPERATIVI

Corso di Laurea in Ingegneria Informatica

A.A. 2019/2020

Ing. Guglielmo Cola


g.cola@iet.unipi.it (specificare "sistemi operativi" nell'oggetto)


www.iet.unipi.it/g.cola/sistemioperativi

ESERCITAZIONE 7

Processi in Unix/Linux (parte III)

Gerarchia di processi – init system

- I sistemi Unix/Linux prevedono un **init** system:
  - Processo mandato in esecuzione dal kernel durante il boot
  - E' il primo processo ad andare in esecuzione (PID = 1)
  - Tutti gli altri processi discendono da init
    - Se un processo termina, gli eventuali figli vengono "adottati" da init
- In Debian/Ubuntu viene utilizzato **systemd** come init system
- Comando per visualizzare l'albero dei processi:
  `pstree`

Identificatori di un processo

- PID ID univoco del processo
- PPID ID del processo padre

- PGID ID del "process group" a cui appartiene il processo

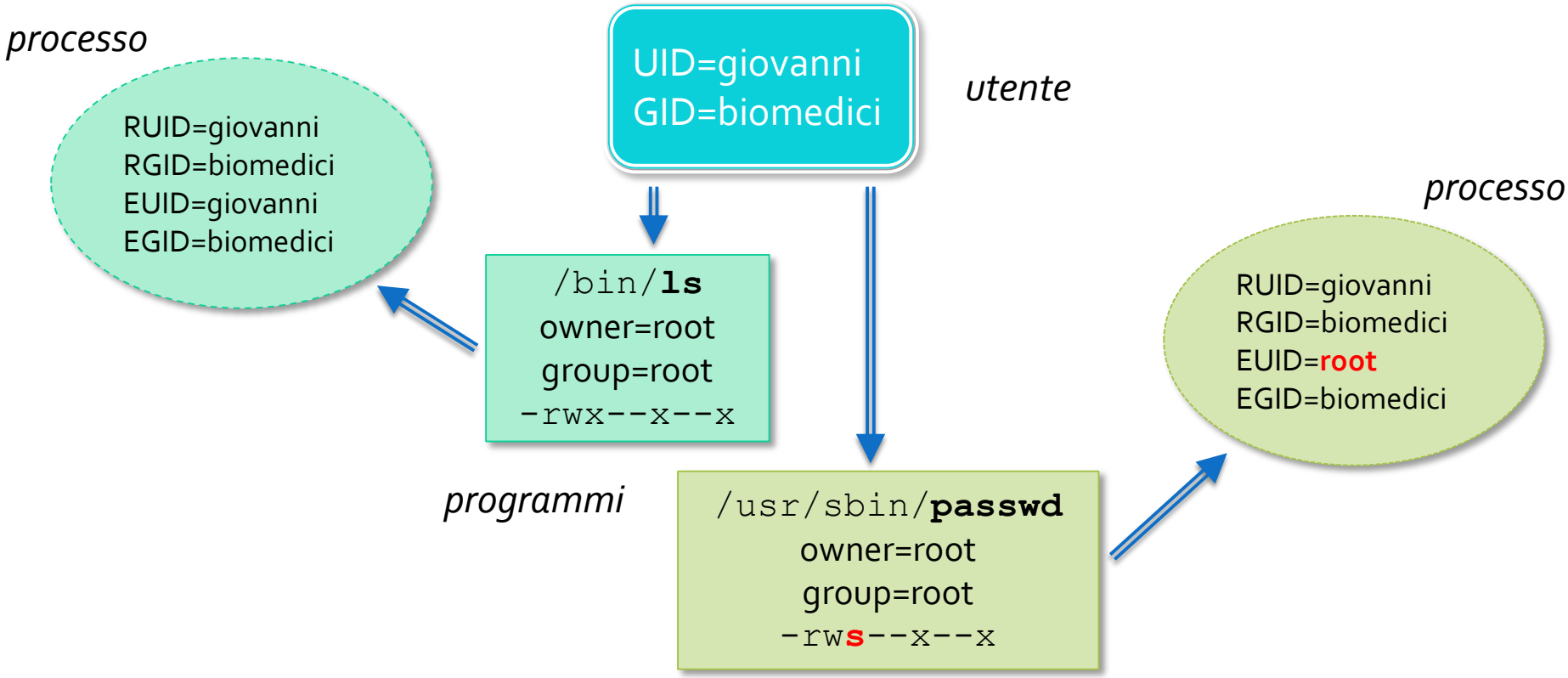
- RUID, RGID Real User/Group ID
- EUID, EGID Effective User/Group ID

Identificatori e permessi

- Gli identificatori che determinano i permessi del processo si dividono in "real" ed "effective"
  - RUID (real user ID) ID dell'utente che ha mandato in esecuzione il processo
  - RGID (real group ID) ID del gruppo primario dell'utente che ha mandato in esecuzione il processo
  - EUID (effective user ID)
  - EGID (effective group ID)

- EUID/EGID possono differire da RUID/RGID se il comando eseguito ha il bit SUID o SGID attivo
  - Vengono utilizzati per definire i privilegi di accesso alle risorse e di invocazione di system call del processo
- Un processo utente (non root) può inviare segnali ad un altro processo solo se il suo EUID o RUID coincide con il RUID del processo destinatario

Identificatori e permessi



Funzioni get per identificatori

- `pid_t getpid()`
- `pid_t getppid()`

- `pid_t getpgrp(void)`

- `uid_t getuid()`
- `uid_t getgid()`
- `uid_t geteuid()`
- `uid_t getegid()`

Gruppi di processi

- I processi sono organizzati in gruppi
  - Quando viene mandato in esecuzione un nuovo processo da terminale, al processo viene associato un nuovo process group
  - Se il processo genera dei figli, questi appartengono allo stesso gruppo
  - Il gruppo viene preservato anche dalla syscall exec
- I gruppi permettono di mandare segnali ad una gerarchia di processi e sono alla base del job-control offerto dalla shell

Priorità dei processi – nice

- Lo scheduler Linux assegna la CPU ai processi tenendo conto di un livello di priorità assegnato a ciascun processo
  - La priorità dipende principalmente dalla classe di scheduling del processo ("real-time" vs "normale")
- La priorità dei processi "normali" può essere in parte controllata mediante il concetto di *niceness* e la relativa system call `nice`
  - Ad ogni processo è associato un valore di niceness nell'intervallo [-20, 19]
  - Un valore di niceness più alto porta ad avere minore priorità di esecuzione
  - In questo modo un processo eseguito in background (non interattivo) può lasciare più tempo di elaborazione agli altri processi
- Solo root può ridurre la niceness di un processo
  - Un utente può solo aumentare la niceness dei suoi processi

GESTIONE DEI PROCESSI DA TERMINALE

(Parte II)

Job-control

- Con job-control si intende la possibilità di sospendere e riattivare gruppi di processi ("jobs") offerta dalla shell mediante opportuni comandi
  - La shell associa un job ID distinto ad ogni comando eseguito
  - Anche a una pipeline di comandi (es. `cat file | grep 'text'`) è associato un solo job
  - I job sono salvati in una tabella specifica, visualizzabile mediante il comando `jobs`

Job-control – foreground vs background

- Un job in esecuzione in **foreground** ha il controllo di standard input, standard output e standard error
  - Di fatto il processo "prende il controllo del terminale" e lo restituisce alla shell alla sua terminazione

- La shell permette anche di eseguire job in **background**

```
comando &
```

  - Il processo non ha più accesso allo standard input
  - L'utente può tornare a utilizzare la shell mentre il job viene completato

Job-control – operazioni sui processi fermati

- Un processo in foreground può essere fermato inviando il segnale `SIGSTP (CTRL+Z)`

- E' possibile intervenire sui job che sono stati fermati in questo modo:
  - Si utilizza `jobs` per ottenere l'identificatore del job (JOB_ID)
  - **`fg`** `JOB_ID` fa ripartire JOB_ID in foreground
  - **`bg`** `JOB_ID` fa ripartire JOB_ID in background

Job-control – kill

- Si può usare il comando kill anche con i job

  `kill %JOB_ID` (invia SIGTERM al job specificato)

  `kill -n SIG %JOB` (invia il segnale SIG)

- Per informazioni su questi comandi di shell utilizzare

  `help nomecomando`

Job-control – disown e nohup

- Se il terminale viene chiuso, i job in esecuzione ricevono il segnale SIGHUP e, di default, vengono terminati
- Ad esempio, se mi connetto in remoto ad un server e lancio dei comandi in background, i rispettivi processi vengono terminati quando mi disconnetto
- Per fare in modo che SIGHUP non porti alla terminazione di un job si possono utilizzare due strumenti
  - `nohup`
  - `disown`

Job-control – nohup

```
nohup comando
```

- Il job eseguito in questo modo è immune a SIGHUP
- Il job non ha più accesso allo stdin
  - In caso di lettura ottiene EOF
- Lo stdout viene rediretto su un file chiamato `nohup.out`

Job-control – disown

```
disown %JOB_ID
```

- Può essere utilizzato per rendere immune a SIGHUP un job già in esecuzione
- Il job viene rimosso dalla tabella dei job, quindi la shell non invierà più il segnale SIGHUP quando viene chiusa
- In questo caso è opportuno fare in modo che il job non legga dallo stdin e che l'eventuale output venga rediretto su file per evitare errori durante l'esecuzione

Comandi nice e renice

- Il comando `nice` permette di mandare in esecuzione un processo con un valore di niceness specificato

```
nice –n valore_nice bzip2 file &
```

- Il comando renice permette di modificare la niceness di un processo già in esecuzione

```
renice valore_nice PID
```

Monitor di sistema – top

- Il comando `top` permette di visualizzare i processi e di effettuare operazioni su di essi in modo interattivo
  - I processi sono ordinati in ordine di utilizzo decrescente della CPU
  - E' possibile inviare segnali ai processi e cambiarne il valore di niceness
  - Vengono visualizzate anche informazioni complessive sul sistema (carico CPU, utilizzo della memoria)

```
top - 00:08:45 up  2:05,  4 users,  load average: 2,05, 2,04, 2,00
Tasks: 139 total,   3 running, 136 sleeping,   0 stopped,   0 zombie
%Cpu(s):100,0 us,  0,0 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   1024372 total,   837496 used,   186876 free,    47764 buffers
KiB Swap:   392188 total,        0 used,   392188 free.   290112 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2596	**studenti**	**20**	**0**	**4076**	**672**	**592**	**R**	**49,3**	**0,1**	**55:50.86**	**a.out**
2597	**studenti**	**20**	**0**	**4076**	**84**	**0**	**R**	**48,9**	**0,0**	**55:50.85**	**a.out**
1257	studenti	20	0	1377272	221144	68112	S	1,3	21,6	0:48.99	gnome-shell
538	root	20	0	299784	84508	19300	S	0,3	8,2	0:13.51	Xorg
1768	www-data	20	0	373400	4488	2856	S	0,3	0,4	0:01.11	apache2
1	root	20	0	110616	4824	3036	S	0,0	0,5	0:00.53	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0,0	0,0	0:00.03	ksoftirqd/0

Monitor di sistema – top

- Esempi di comandi interattivi
  - h help
  - d intervallo di aggiornamento (delay)
  - k invio di un segnale
  - n numero di processi da visualizzare
  - r renice
  - u utente da visualizzare
  - q quit

ESERCIZI

Esercizio 1 (1/2)

- Scrivere un programma C in cui
  - Viene creato un processo figlio
  - Padre e figlio entrano in un loop infinito
- Mandare in esecuzione il programma in background
- Utilizzare i comandi del job-control per far tornare il job in foreground
- Fermare (SIGTSTP) il job da tastiera
- Usare il comando "`ps o comm,user,pid,ppid,pgid`"
  - Chi è il padre del processo padre?
  - Padre e figlio hanno lo stesso PGID?

Esercizio 1 (2/2)

- Utilizzare i comandi del job-control per far ripartire il job in background
- Utilizzare il comando `disown` per fare in modo che il job diventi immune a SIGHUP
  - Usare il comando `jobs` e verificare che il job non è più nella lista
  - Chiudere il terminale e aprire un nuovo terminale
  - Utilizzare il comando `ps` come prima. I due processi (padre e figlio) vengono visualizzati? Quale opzione è necessario aggiungere al comando ps per vederli?
  - Qual è adesso il PPID del processo padre?
- Aprire il manager di sistema `top`
  - Impostare l'intervallo di aggiornamento (delay) a un secondo
  - Inviare il segnale SIGTERM ai due processi per terminarli

Esercizio 2 (1/2)

- Creare un archivio tar (non compresso) con il contenuto di /usr/bin
- Comprimere l'archivio con `gzip`
  - Utilizzare l'opzione che permette di mantenere l'archivio originale, altrimenti agli step successivi sarà necessario ricreare l'archivio di partenza
  - Utilizzare il comando time (time gzip…) che stampa il tempo impiegato dal processo quando termina l'esecuzione
- Ripetere la gzip, stavolta con `nice -n 19`. Ci mette più tempo di prima?

Esercizio 2 (2/2)

- Mandare in esecuzione (in background) il programma C realizzato nell'esercizio precedente
- Eseguire nuovamente `gzip` con `nice -n 19`.
  Cosa succede?
  - Aprire un altro terminale e lanciare il comando `top` per controllare quanta CPU viene assegnata al processo
- Terminare `gzip` (se non ha ancora finito) e stavolta eseguire il comando con `nice -n 5`
  - Controllare su top come viene suddivisa la CPU tra i vari processi