Gestione dei Processi

Licenza di uccidere

Stefano Bouchs
dr.schteeven@gmail.com

Corsi Linux Avanzati 2014

POuL
have a lot of fun...

I processi

Definizione

I processi sono l'astrazione di un programma in esecuzione che ci permette di gestire e monitorare l'utilizzo delle risorse.

Informazioni sui processi

Ai processi sono collegate tutta una serie di informazioni sul programma in esecuzione:

- Address Space[1]
- Stato del processo
- Priorità
- Risorse in uso
- Proprietario

[1] Codice, Dati

Attributi di un processo

- PID (Process Identifier): Numero univoco che identifica ogni processo.
- PPID (Parent Process Identifier): PID del processo padre.
- UID (User ID): Identificatore dell'utente che ha creato il processo. [2]
- EUID (Effective UID): UID effettivo. Indica per quali risorse il processo possiede i permessi.
- GID ed EGID: Stesso contetto di UID e EUID ma applicato ai gruppi.
- COMMAND: Riga di comando del processo lanciato.

[2] Di solito è la copia dell'UID del padre.

Creazione di un processo

In Linux tutti i processi si "riproducono per mitosi".

- Il processo padre crea il figlio utilizzando la chiamata di sistema `fork();`
- La `fork();` resituisce 0 al processo figlio ed il PID del figlio al processo padre.
- Il valore della fork viene valutato da entrambi ed eseguono porzioni di codice differenti dello stesso programma.
- Il figlio può cambiare il proprio codice attravero una delle chiamate `exec();`

Morte di un processo

Quando un processo termina la sua esecuzione esegue una `exit();`

- La chiamata ritorna un valore di uscita che deve essere raccolto dal padre (se ancora vivo).
- Quando usiamo il terminale ogni comando genera un figlio che esegue il programma
- Possiamo vedere il codice di uscita del comando lanciato con `echo $?`
- `&&` e `||` ci permettono di eseguire comandi concatenati in base al codice di uscita
  - `apt-get update && apt-get upgrade`

Init

"Init è il padre di tutti i processi."

- Primo processo creato dal sistema, unico senza padre[3]
- PID=1
- Agisce da "padre adottivo" per qualunque processo "orfano"
  - Se il padre muore prima del figlio, init si occupa di raccogliere lo stato del figlio alla fine della sua esecuzione

[3] Maggiori info nella prossima lezione

Stati dei processi

Durante la loro esistenza i processi posso attraversare diversi stati:

- Runnable [R]: Pronto, in attesa di esecuzione.
- Sleeping [S/D]: In attesa di uno specifico evento.
- Zombie [Z]: L'esecuzione del processo è conclusa ma nessuno ha ancora raccolto lo stato.
- Stopped [T]: Fermato da un segnale.

Background

I processi in background sono processi generati da una shell che sono stati messi in secondo piano

- `[Comando] &`
- `nohup [Comando] &` permette al processo di esistere ancora nonostante la chiusura del terminale[4]
- Per mettere un processo già in esecuzione in background è necessario "stoppare" (CTRL +Z) il processo e lanciare il comando `bg`
- `jobs` mostra il processi in background di una shell
- `fg` rimette in foreground l'ultimo processo messo in background (a meno di non specificare un altro job id)
- Per avere più terminali "virtuali" senza mettere i processi esplicitamente in background si usa `screen`
  - Possiamo creare delle sessioni e lasciarle aperte mentre facciamo altro
  - Utile per terminali remoti dove la connessione può cadere

[4] No HUP, vedi slide "Segnali Famosi"

I segnali

I segnali sono delle richieste di interrupt particolari che modificano lo stato di un processo:

- Identificati con numeri interi tra 1 e 64
- Da 1 a 31 segnali standard da 32 a 64 segnali real-time
- Possono essere usati dai processi per comunicare ed interagire tra loro.
- Possono essere inviati da terminale con combinazioni di tasti (CTRL + C o CTRL + Z)
- Possono essere inviati dal Kernel
- Soprattutto possono essere inviati dall'Admin

Gestione dei segnali

- Se il processo ha una routine per gestire il segnale essa viene chiamata, altrimenti se ne occupa il Kernel
- I programmi possono richiedere che alcuni segnali vengano ignorati o bloccati[5]
  - Se ignorati vengono semplicemente cestinati
  - Se bloccati rimangono in coda, ma la routine di gestione viene chiamata una sola volta

[5] Visibile in `/proc/[PID]/status`

Segnali famosi[7]

- STOP: Ferma l'esecuzione di un processo
- CONT: Riprende l'esecuzione di un processo dopo STOP
- TSTP: Simile a stop ma inviato con CTRL +Z da terminale (`fg` per sbloccarlo)
- HUP [1]: Di solito viene usato per ricaricare la configurazione nei demoni (server web) o inviato dalla shell a programmi in background
- INT [2]: Richiesta di terminare l'operazione corrente (CTRL + C) solitamente risulta nella morte "pulita" del processo
- TERM [15]: Segnale di arresto "gentile", ci si aspetta che il processo gestisca la propria morte
- KILL [9]: Segnale non bloccabile, gestito dal Kernel che scarica dalla memoria il programma. Non viene mai ricevuto in realtà[6]
- QUIT [3]: Simile a term ma produce un core dump se non raccolto
- USR1/USR2: User-defined, usati spesso per il reload dei file di configurazione

[6] Garantisce la morte del processo ma non è sempre immediata (DMA o IO in corso)

[7] I nomi seguenti sono di solito preceduti da SIG

Inviare segnali

Per inviare segnali ad un processo si utilizza il comando kill

- `kill [-segnale] PID`
- Se non specificato invia il segnale TERM

Esempio:

```
kill -9 1337
kill -SIGKILL 1337
```

Inviare segnali 2 (Falsi amici)

Il comando killall si comporta in maniera differente su Linux rispetto ad Unix

- Su Linux uccide i processi per nome
  - `killall [-segnale] [Nome programma]`
- Su Unix uccide tutti i processi dell'utente
  - Se root uccide init

Nice

Nice: valore che determina la “gentilezza” di un processo nel cedere le risorse agli altri

- Valore compreso tra -20 e 19
- Nice alto = bassa priorità; Nice basso = alta priorità
- Ereditato dal padre
- Può essere alzato dal proprietario del processo ma non abbassato
- Solo l’amministratore può abbassare il nice

“Nice processes finish last.”

Nice e Renice

Per avere un valore diverso da quello ereditato ci sono due modi:

- Si avvia il programma con il comando `nice` specificando il valore desiderato
  - Esempio: `nice -n 5 ~/bin/zerbino`
- Si modifica durante l'esecuzione attraverso il comando `renice`[8]
  - Esempio: `sudo renice -5 1337`

Un utilizzo pratico è quello di lanciare una shell con nice basso per indagare lo stato di una macchina sotto carico eccessivo.

[8] Con permessi di amministratore

/proc

`/proc` è uno pseudo file-system dove si trovano tutte le informazioni sui processi e parametri del sistema operativo

- Tutti i successivi comandi prendono le informazioni da qui
- Cartelle con info generali
  - `/proc/cpuinfo` , `/proc/meminfo`
- Cartelle per ogni processo (PID)
  - `cwd` link simbolico alla cartella di lavoro del processo
  - `cmdline` contiene la stringa con cui è stato lanciato il programma
  - `fd` contiene file numerati con i file descriptor aperti

Monitorare i processi

`ps` : mostra informazioni sui processi (PID,UID,Nice,tty,memoria,stato, etc)

- Principalemente pensato per sviluppatori, non sysadmin
- Uso frequente: `ps aux`
  - `a` : mostra tutti i processi
  - `u` : formato utente
  - `x` : mostra anche processi senza terminale di controllo
- Non si refresha, stampa solo a schermo le informazioni
- Usato assieme a `grep` per cercare informazioni dettagliate

Monitorare la memoria del sistema

- `free` ci permette di avere informazioni sulla quantità di memoria di sistema libera o in uso
- Il parametro `-m` per visualizzare in MegaByte `-g` per i GigaByte
- Riporta anche informazioni sullo swap
  - Lo swap è una partizione su disco che viene utilizzata quando la memoria reale non basta a contenere i processi
    - `swapon -s` per sapere le partizioni swap abilitate
    - Abilitare/Disabilitare swapon swapoff

Monitorare i processi in tempo reale™

`top`

- Refresh ogni 3s
- Può mandare segnali e modificare il nice dei processi
- `-u` processi di un solo utente

`htop`

- Versione figa "colorosa" di `top`
- Non installato di default ma sempre disponibile nei repo

Osservare i processi

`strace`: si aggancia ad un processo e mostra ogni chiamata di sistema eseguita e segnale ricevuto

- `strace -p [PID]`
- `-f` segue le fork

`lsof` ci permette di vedere tutti i file aperti:

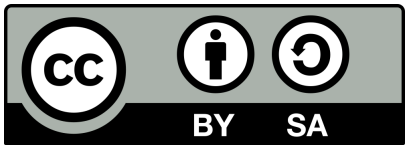
- `lsof -p [PID]`
- Poiché ogni cosa è un file possiamo avere molte informazioni sul comportamento di un processo
- `lsof -n` non risolve gli indirizzi ip dei socket aperti
- Uso frequente: Capire quale processo sta usando ancora una chiavetta usb che vogliamo rimuovere

Fonti

- Unix and Linux System Administration Handbook (Fourth Edition)
- Slides degli anni precedenti di Radu Andries (Corsi Avanzati 2013), di Pietro Virgilio (Corsi Avanzati 2012) e di Daniele Iamartino (Corsi Avanzati 2011)
- Google && Stackexchange ;-)

Addio e grazie per tutto il pesce

Grazie per l'attenzione!



`http://www.poul.org`