Gestione dei processi

In questo capitolo imparerai come lavorare con i processi.

Obiettivi : In questo capitolo, futuri amministratori Linux impareranno come:

✔ Riconoscere il `PID` e il `PPID` di un processo;
✔ Visualizzare e cercare processi;
✔ Gestire i processi.

🏁 **processi**, **linux**

Conoscenza: ⭐⭐
Complessità: ⭐

Tempo di lettura: 20 minuti

Generalità

Un sistema operativo è costituito da processi. Questi processi sono eseguiti in un ordine specifico e sono correlati tra loro. Ci sono due categorie di processi, quelli focalizzati sull'ambiente utente e quelli focalizzati sull'ambiente hardware.

Quando viene eseguito un programma, Il sistema creerà un processo posizionando i dati del programma e il codice in memoria e creando una **runtime stack**. Un processo è quindi un'istanza di un programma con un ambiente di processore associato (contatore ordinale, registri, etc...) e ambiente di memoria.

Ogni processo ha:

- un ***PID*** : ***Process IDentifier***, un identificatore di processo unico;
- un ***PPID*** : ***Parent Process IDentifier***, identificatore univoco del processo genitore.

Da filiazioni successive, il processo `init` è il padre di tutti i processi.

- Un processo è sempre creato da un processo genitore;
- Un processo genitore può avere più processi figlio.

C'è una relazione genitore/figlio tra i processi. Un processo figlio è il risultato del processo

genitore che chiama il ***fork ()*** iniziale e duplicando il proprio codice crea un processo figlio. Il ***PID*** del processo figlio viene restituito al processo genitore in modo che possa comunicare. Ogni processo figlio ha l'identificatore del suo processo genitore, il ***PPID***.

Il numero ***PID*** rappresenta il processo al momento dell'esecuzione. Quando il processo finisce, il numero è di nuovo disponibile per un altro processo. Eseguendo lo stesso comando più volte produrrà un diverso ***PID*** ogni volta.!!! Note "Nota"

```
I processi non devono essere confusi con i _threads_. Ogni processo ha il
proprio contesto di memoria (risorse e spazio di indirizzamento), mentre il
threads dello stesso processo condivide lo stesso contesto.
```

Visualizzazione dei processi

Il comando `ps` visualizza lo stato dei processi in esecuzione.

```
ps [-e] [-f] [-u login]
```

Esempio:

```
# ps -fu root
```

Opzione	**Descrizione**
`-e`	Visualizza tutti i processi.
`-f`	Visualizza ulteriori informazioni.
`-u` login	Visualizza i processi dell'utente.

Alcune opzioni aggiuntive:

Opzione	**Descrizione**
`-g`	Visualizza i processi nel gruppo.
`-t tty`	Visualizza i processi in esecuzione dal terminale.
`-p PID`	Visualizza le informazioni del processo.
`-H`	Visualizza le informazioni in una struttura ad albero.

Opzione	Descrizione
`-I`	Visualizza ulteriori informazioni.
`--sort COL`	Ordina il risultato secondo una colonna.
`--headers`	Visualizza l'intestazione su ogni pagina del terminale.
`--format "%a %b %c"`	Personalizza il formato di visualizzazione dell'uscita.

Senza un'opzione specificata, il comando `ps` visualizza solo i processi in esecuzione sul terminale corrente.

Il risultato viene visualizzato in colonne:

```
# ps -ef
UID  PID PPID C STIME  TTY TIME      CMD
root 1   0    0 Jan01  ?   00:00/03  /sbin/init
```

Colonna	Descrizione
`UID`	Utente proprietario.
`PID`	Identificatore di processo.
`PPID`	Identificatore del processo genitore.
`C`	Priorità del processo.
`STIME`	Data e ora di esecuzione.
`TTY`	Terminale di esecuzione.
`TIME`	Durata di elaborazione.
`CMD`	Comando eseguito.

Il comportamento del controllo può essere completamente personalizzato:

```
# ps -e --format "%P %p %c %n" --sort ppid --headers
 PPID   PID COMMAND          NI
```

```
    0     1 systemd          0
    0     2 kthreadd         0
    1   516 systemd-journal  0
    1   538 systemd-udevd    0
    1   598 lvmetad          0
    1   643 auditd          -4
    1   668 rtkit-daemon     1
    1   670 sssd             0
```

Tipi di processi

Il processo dell'utente:

- è iniziato da un terminale associato a un utente;
- accede alle risorse tramite richieste o daemons.

Il processo di sistema (***daemon***):

- è iniziato dal sistema;
- non è associato a nessun terminale, ed è di proprietà di un utente di sistema (spesso `root`);
- è caricato al momento dell'avvio, risiede in memoria, e sta aspettando una chiamata;
- è solitamente identificato dalla lettera `d` associato al nome del processo.

I processi di sistema sono quindi chiamati daemons (***Disk And Execution MONitor***).

Autorizzazioni e diritti

Quando viene eseguito un comando, le credenziali dell'utente sono passate al processo creato.

Per impostazione predefinita., l'attuale `UID` e `GID` (del processo) sono quindi identici al **effettivo** `UID` e `GID` (il `UID` e `GID` dell'utente che ha eseguito il comando).

Quando un `SUID` (e/o `SGID`) è impostato su un comando, l'attuale `UID` (e/o `GID`) diventa quello del proprietario (e/o gruppo proprietario) del comando e non più quello dell'utente o del gruppo di utenti che ha emesso il comando. Effettivo e reale **UIDs** sono quindi **differenti**.

Ogni volta che si accede a un file, il sistema controlla i diritti del processo in base ai suoi effettivi identificatori.

Gestione dei processi

Un processo non può essere eseguito indefinitamente, perchè questo sarebbe a discapito di altri processi in esecuzione e impedirebbe il multitasking.

Il tempo totale di elaborazione disponibile è quindi diviso in piccoli intervalli, e ogni processo (con una priorità) accede al processore in modo sequenziale. Il processo prenderà diversi stati durante la sua vita tra gli stati:

- pronto: in attesa della disponibilità del processo;
- in esecuzione: accede al processore;
- sospeso: aspettando un I/O (input/output);
- fermato: aspettando un segnale da un altro processo;
- zombie: richiesta di distruzione;
- morto: il padre del processo chiude il suo processo figlio.

La sequenza di chiusura del processo è la seguente:

1. Chiusura dei file aperti;
2. Rilascio della memoria usata;
3. Invio di un segnale ai processi genitore e figlio.

Quando un processo genitore muore, si dice che i suoi processi figli sono orfani. Sono quindi adottati dal processo `init` che li distruggerà.

La priorità di un processo

Il processore funziona in condivisione del tempo (time sharing) con ogni processo occupando una determinata quantità di tempo del processore.

I processi sono classificati per priorità il cui valore varia da **-20** (la massima priorità) a **+19** (la priorità più bassa).

La priorità predefinita di un processo è **0**.

Modalità di funzionamento

I processi possono essere eseguiti in due modi:

- **sincrona**: l'utente perde l'accesso alla shell durante l'esecuzione del comando. Il prompt dei comandi riappare alla fine dell'esecuzione del processo.
- **asincrona**: il processo viene elaborato in background. Il prompt dei comandi viene visualizzato di nuovo immediatamente.

I vincoli della modalità asincrona:

- il comando o lo script non devono attendere l'input della tastiera;
- il comando o lo script non devono restituire alcun risultato sullo schermo;

- lasciare che la shell termini il processo.

Controlli per la gestione dei processi

comando `kill`

Il comando `kill` invia un segnale di arresto a un processo.

```
kill [-signal] PID
```

Esempio:

```
$ kill -9 1664
```

Codice	Segnale	Descrizione
`2`	***SIGINT***	Arresto immediato del processo
`9`	***SIGKILL***	Interruzione del processo (CTRL + D)
`15`	***SIGTERM***	Arresto pulito del processo
`18`	***SIGCONT***	Riprendere il processo
`19`	***SIGSTOP***	Sospendere il processo

I segnali sono i mezzi di comunicazione tra i processi. Il comando `kill` invia un segnale a un processo.

> **Suggerimento**
>
> L'elenco completo dei segnali presi in considerazione dal comando `kill` è disponibile digitando il comando :
>
> ```
> $ man 7 signal
> ```

comando `nohup`

`nohup` consente il lancio di un processo indipendentemente da una connessione.

```
comando nohup
```

Esempio:

```
$ nohup myprogram.sh 0</dev/null &
```

`nohup` ignora il segnale `SIGHUP` inviato quando un utente si disconnette.

Domanda

`nohup` gestisce l'output standard e l'errore, ma non l'input standard, quindi il reindirizzamento di questo input a `/dev/null`.

[CTRL] + [Z]

Premendo la combinazione `CTRL` + `Z` contemporaneamente, il processo sincrono è temporaneamente sospeso. L'accesso al prompt viene ripristinato dopo aver visualizzato il numero del processo che è stato appena sospeso.

istruzione `&`

La dichiarazione `&` esegue il comando in modo asincrono (il comando viene quindi chiamato ***job***) e visualizza il numero di ***job***. L'accesso al prompt viene quindi restituito.

Esempio:

```
$ time ls -lR / > list.ls 2> /dev/null &
[1] 15430
$
```

Il numero ***job*** è ottenuto durante l'elaborazione in background e viene visualizzato in parentesi quadre, seguito dal numero di `PID`.

comandi `fg` e `bg`

Il comando `fg` mette il processo in primo piano:

```
$ time ls -lR / > list.ls 2>/dev/null &
$ fg 1
time ls -lR / > list.ls 2/dev/null
```

mentre il comando `bg` lo colloca in background:

```
[CTRL]+[Z]
^Z
[1]+ Stopped
$ bg 1
[1] 15430
$
```

Se è stato messo in background quando è stato creato con l'argomento `&` o più tardi con la combinazione `CTRL` + `Z`, un processo può essere riportato in primo piano con il comando `fg` e il suo numero di lavoro.

comando `jobs`

Il comando `jobs` visualizza l'elenco dei processi in esecuzione in background e specifica il loro numero di lavoro.

Esempio:

```
$ jobs
[1]- Running    sleep 1000
[2]+ Running    find / > arbo.txt
```

Le colonne rappresentano:

1. numero di lavoro;
2. l'ordine in cui i processi sono in esecuzione
3. un `+` : questo processo è il prossimo processo da eseguire per impostazione predefinita con `fg` o `bg` ;
4. un `-` : questo processo è il prossimo processo a prendere il `+` ;
5. ***Running*** (processo in esecuzione) o ***Stopped*** (processo sospeso).
6. il comando

comandi `nice` e `renice`

Il comando `nice` consente l'esecuzione di un comando specificando la sua priorità.

```
comando nice priority
```

Esempio:

```
$ nice -n+15 find / -name "file"
```

a differenza di `root`, un utente standard può solo ridurre la priorità di un processo. Saranno accettati solo valori tra +0 e +19.

> **Suggerimento**
>
> Quest'ultima limitazione può essere eliminata per utente o per gruppo modificando il file `/etc/security/limits.conf`.

Il comando `renice` ti consente di modificare la priorità di un processo di esecuzione.

```
renice priority [-g GID] [-p PID] [-u UID]
```

Esempio:

```
$ renice +15 -p 1664
```

| Opzione | Descrizione | | ------- | -- | | `-g` | `GID` del gruppo proprietario del processo. | | `-p` | `PID` del processo. | | `-u` | `UID` del proprietario del processo. |

Il comando `renice` agisce sui processi già in esecuzione. È quindi possibile modificare la priorità di un processo specifico, ma anche di diversi processi appartenenti a un utente o un gruppo.

> **Suggerimento**
>
> Il comando `pidof`, associato al comando `xargs` (vedi il corso Comandi avanzati), permette di applicare una nuova priorità in un singolo comando:
>
> ```
> $ pidof sleep | xargs renice 20
> ```

comando `top`

Il comando `top` visualizza i processi e il loro consumo di risorse.

```
$ top
PID  USER PR NI ... %CPU %MEM  TIME+    COMMAND
2514 root 20 0       15    5.5 0:01.14   top
```

Colonna	Descrizione
`PID`	Identificatore del processo.
`USER`	Utente proprietario.
`PR`	Priorità del processo.
`NI`	Valore di Nice.
`%CPU`	Carico del processore.
`%MEM`	Carico di memoria.
`TIME+`	Tempo di utilizzo del processore.
`COMMAND`	Comando eseguito.

Il comando `top` consente il controllo dei processi in tempo reale e in modalità interattiva.

comandi `pgrep` e `pkill`

Il comando `pgrep` cerca i processi in esecuzione per un nome di processo e visualizza il ***PID*** che soddisfa i criteri di selezione sull'output standard.

Il comando `pkill` invierà il segnale specificato (per impostazione predefinita ***SIGTERM***) ad ogni processo.

```
pgrep process
pkill [-signal] process
```

Esempi:

- Ottenere il numero di processo di `sshd`:

```
$ pgrep -u root sshd
```

- Termina tutti i processi `tomcat`:

```
$ pkill tomcat
```

Ultimo aggiornamento: 5 gennaio 2022