

Eeguire un processo su Linux

In un sistema UNIX, un **processo è fondamentalmente equivalente a un'applicazione in esecuzione**. La gestione dei processi è relativamente semplice e diretta. I comandi che utilizziamo sono in realtà programmi che risiedono nel **file system**; quindi, quando parliamo di comandi, ci riferiamo a varie applicazioni collocate in specifiche directory del sistema. Quando desideriamo eseguire un comando, il sistema operativo cerca il programma corrispondente all'interno delle cartelle designate nel file system. Queste directory sono generalmente specificate in una **variabile d'ambiente** chiamata `PATH`, che elenca i percorsi in cui il sistema cercherà i comandi. Se il comando è presente in una delle cartelle indicate, verrà eseguito come un processo. Altrimenti, riceveremo un messaggio di errore che indica che il comando non è stato trovato. In questo modo, la gestione e l'esecuzione dei processi su UNIX avviene in maniera chiara e strutturata.

Il comando che utilizziamo per eseguire un programma è il seguente:

```
./applicazione_da_eseguire
```

Possiamo unire diversi comandi all'interno di un unico **script** per poterli eseguire tutti insieme in una sola volta.

Per scriverli basta usare **l'estensione .sh**, mentre per eseguirli:

```
Bash script.sh // esegue lo script
For Bash script #1/bin/user //Leggera il carattere speciale #1 e lo compila
con python
```

Possiamo inoltre eseguire i vari processi in due modi distinti:

- **Foreground:** L'applicazione ha il **controllo della riga di comando**.
- **Background;** Mi permette di riprendere controllo della bash

Comandi per mandare in background/foreground:

```
comando & //Esegua il comando e lo manda in background
fg //Manda un foreground un processo sospeso
bg //Manda in background un processo sospeso
```

Quando chiamo un programma in background l'**output finisce sullo standard output** il che da fastidio perchè il controllo c'è l'ha l'applicazione, quindi posso mandare il risultato in un file di output.

jobs - comando per capire quanti processi ci sono. Le chiama jobs perché non distingue da script o altro.


[1] - significa che il job è stato mandato in background ed è stato finito.

Esistono inoltre una serie di comandi con i quali posso mandare dei segnali ai processi:

- **CTRL-C**: **uccide un processo**, In ogni applicazione esiste un Handler attivata dal comando di CTRL-C inserita nella fase del linking da gcc/g++. CTRL-C è un segnale mandato **dalla bash(processo1) al processo2**.
- **CTRL-Z**: **Blocca un processo**.
- **CTRL-A**: **esco dalla screen**.

Differenza tra segnali ed interrupt

- Segnali: Inviati da un processo all'altro
- Interrupt: Causati dall'hardware (di genere legati all'I/O)

 **Kill() non è uccidere è mandare un segnale, la terminazione del segnale è solo uno dei possibili segnali.**

Il comando **exit()** **termina la bash** e per terminarla devo chiudere tutti i processi.

Per uccidere un processo devo fare kill PID.

ATTENTO A NON INSERIRE IL NUMERO DEL PROCESSO.

Se dobbiamo uccidere un processo con figli **conviene uccidere prima il padre e poi i figli**.

Comandi NOHUP e SCREEN

nohup consente di **eseguire un processo in modo che continui a funzionare anche dopo che l'utente ha disconnesso la sessione**. Il nome "nohup" sta per "no hang up", riferendosi al fatto che il processo non viene interrotto quando si chiude il terminale.

Quando si utilizza **nohup**, l'output standard e l'output di errore vengono reindirizzati a un file chiamato **nohup.out**, a meno che non venga specificato diversamente. È particolarmente utile per eseguire script o processi di lunga durata in background

screen è un **terminal multiplexer** in Linux che consente di **eseguire più sessioni di terminale all'interno di una singola finestra del terminale**. È particolarmente utile per gestire processi di

lunga durata, poiché consente di scollegarsi da una sessione senza interrompere i processi in esecuzione.

Creazione di un Nuovo Utente su Linux

Motivazioni per Non Usare l'Utente Amministratore

1. **Sicurezza:** Se l'utenza viene compromessa, un attaccante può ottenere il controllo totale della macchina.
2. **Prevenzione degli Errori:** Un comando errato eseguito come amministratore può danneggiare gravemente il sistema.

Creazione di un Nuovo Utente

Utilizzare il comando `sudo` per creare un nuovo utente:

```
sudo useradd -s /path/to/shell -d /home/{dirname} -m -G {secondary-group} {username}
```

Parametri del comando:

- `-s` : Specifica la shell dell'utente.
- `-d` : Indica la directory home dell'utente.
- `-m` : Crea la directory home se non esiste.
- `-G` : Aggiunge l'utente a gruppi secondari.
- `{username}` : Nome dell'utente da creare.

Assegnazione della Password

Per impostare una password per il nuovo utente, utilizzare:

```
sudo passwd {username}
```

Le password sono memorizzate nel file `/etc/shadow`.

Visualizzazione del File Shadow

Per visualizzare il contenuto del file `/etc/shadow`, puoi usare:

```
sudo less /etc/shadow
```

Crittografia delle Password

- **Algoritmi di Cifratura Lenti:** Vengono utilizzati per rendere più difficile l'attacco di brute force, poiché richiedono più tempo per calcolare.
- **Funzioni Hash:** Queste trasformano una stringa in un valore univoco, rendendo impossibile risalire alla stringa originale.

Considerazioni sulla Sicurezza

Anche se le funzioni hash possono essere vulnerabili se si conosce l'algoritmo utilizzato, l'uso di un'utenza limitata riduce il potenziale danno in caso di compromissione.

Crittografia Asimmetrica

Quando si crea un'utenza, si può considerare l'uso della crittografia asimmetrica (chiave pubblica e privata) per migliorare ulteriormente la sicurezza.