Corsi Linux 2018

Processi

Domenico Iezzi

October 9, 2018

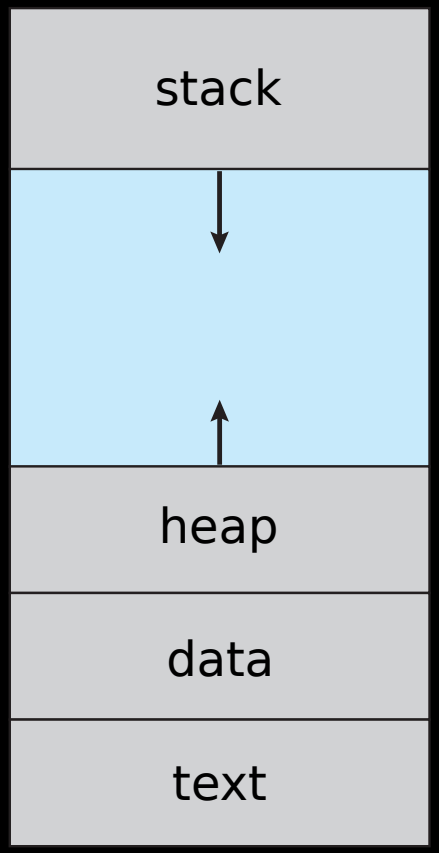
POLITECNICO OPEN
unix LABS

Cos'è un processo

Un processo è un istanza di un programma in esecuzione. È un entità *attiva* che consiste in:

- Il codice del programma (**text section**)
- Attività (**program counter** e **registers**)
- Dati (**stack**, **heap** e **data section**)

N.B.
Un programma è un entità *passiva*, ossia un set di istruzioni e dati contenuti in un file su disco.



Proprietà

- PID: identificatore univoco del processo. Numero intero non negativo da 1 ad un massimo di 32767, assegnato in maniera sequenziale
- PPID: identificatore del processo che ha generato il processo corrente
- UID e GID: user e group identifier stabiliscono quali risorse un determinato processo può accedere
  - RUID: proprietario del processo
  - EUID: usato per controllare gli accessi. File generati da un processo erediteranno questo ID
- TTY: terminale che esegue il processo
- ...

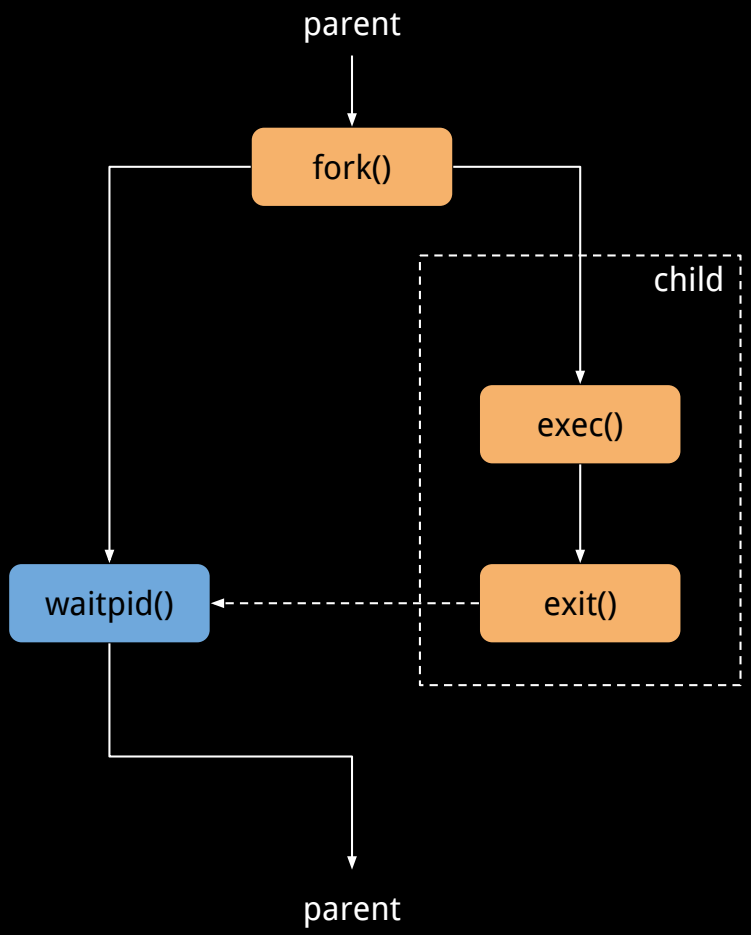
Process lifecycle

Un processo viene creato tramite le system call fork + exec.

- fork crea un processo figlio come una copia esatta del processo padre, con un PID diverso.
- exec sostituisce il processo corrente con un nuovo programma, che verrà eseguito immediatamente

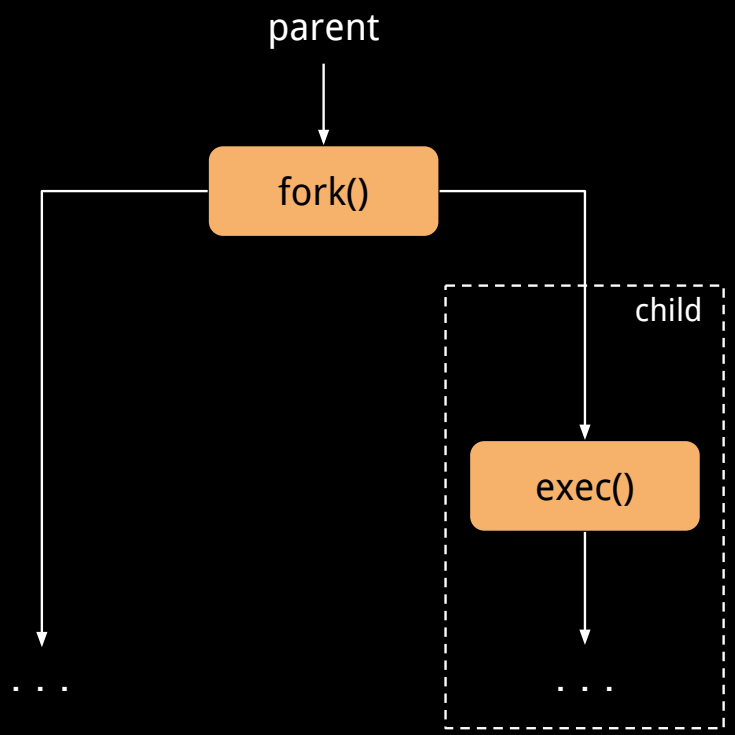
Process lifecycle

Il processo padre può rimanere in attesa del termine dell'esecuzione del figlio...



Process lifecycle

...oppure continuare l'esecuzione



Exit status

Ogni processo passerà al processo padre un valore al termine dell'esecuzione

- Un numero pari a 0 indica che l'esecuzione è andata a buon fine
- Numeri da 1 a 255 vengono utilizzati per indicare un errore nell'esecuzione del programma
- Il significato di questi valori cambia di programma in programma, ed in genere indica la causa specifica di errore e.g. in bash 127 == *command not found*

Special processes

- Un processo diventa orfano quando il parent termina la sua esecuzione
- Un processo diventa zombie quando ha terminato l'esecuzione e rimane una entry nel *process table*
- Un daemon è un processo eseguito in background non controllato da un terminale e.g. sshd, systemd-logind, upowerd ...
- Il daemon init è il capostipite di tutti i processi
  - non ha un parent (orfano di default)
  - PID uguale a 1
  - adotta tutti i processi orfani
- Un processo può diventare daemon utilizzando la tecnica del *double fork*

Signals

I segnali sono delle notifiche asincrone inviate ad un processo in occasione di particolari eventi. Un processo può intercettare determinati segnali per poter fornire una routine personalizzata di gestione

- SIGINT: richiede l'interruzione di un processo e.g. Ctrl-C per interrompere un processo in bash
- SIGKILL: richiede la terminazione immediata del processo, non può essere ignorato o intercettato
- SIGTERM: simile a SIGKILL ma può essere ignorato o intercettato da un processo per poter terminare in maniera corretta
- SIGSTOP e SIGCONT: stop e resume di un processo
- SIGHUP: inviato ad un processo quando il terminale che lo controlla viene chiuso

Process management: ps

`ps` è utilizzato per stampare informazioni sui processi attivi. Di default stamperà pid, tty e tempo di esecuzione dei processi relativi alla sessione della shell corrente. È possibile specificare altre opzioni, tra cui:

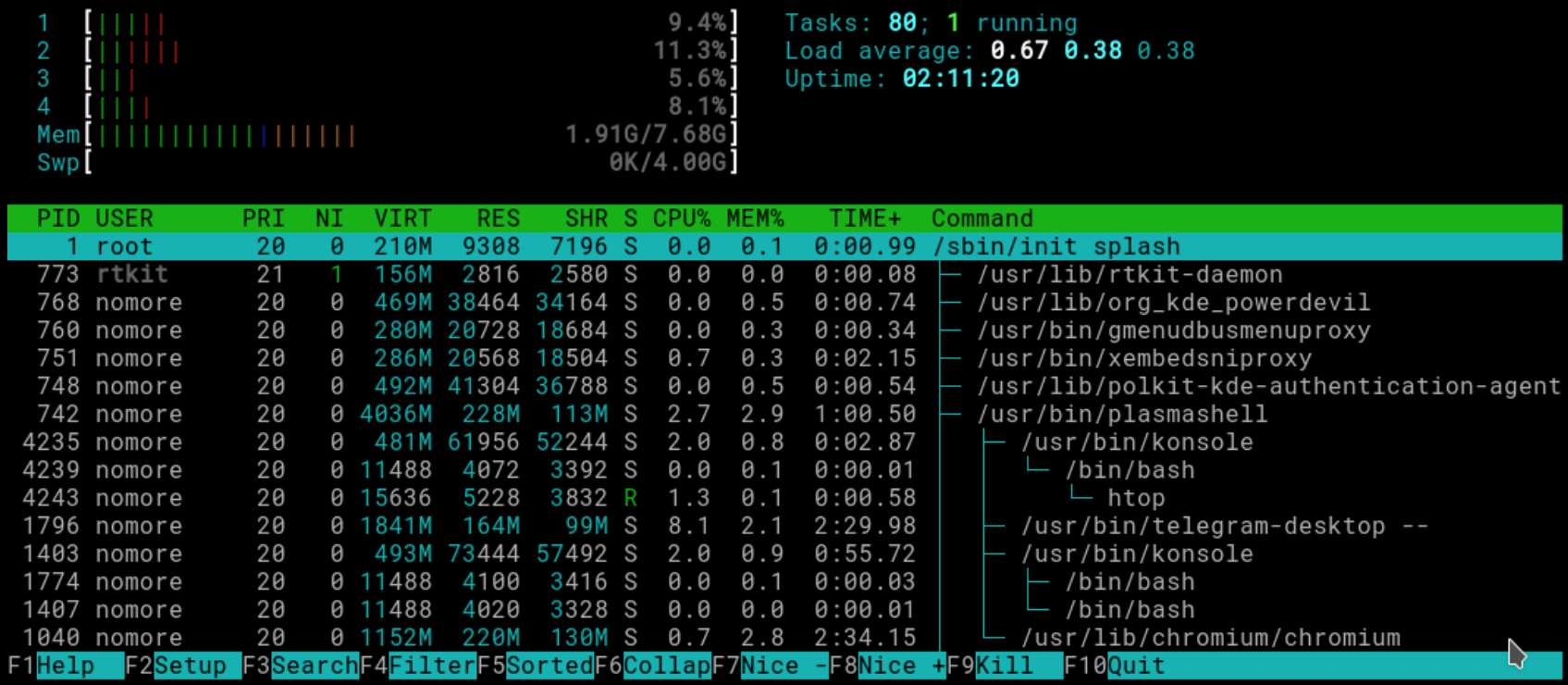
- a visualizza processi di tutti gli utenti
- x visualizza tutti i processi anche se non hanno un terminale di controllo
- u visualizza maggiori informazioni in un formato più adatto alla lettura da parte di un utente
- -o format permette di personalizzare le proprietà da visualizzare

Esempio:

```
$ ps -o pid,ppid,tty,args a
  PID  PPID TT       COMMAND
  541   539 tty1     /usr/lib/Xorg
 1407  1403 pts/1    /bin/bash
```

Process management: top/htop

`top` permette di vedere statistiche del sistema e la lista di processi attivi in maniera interattiva. `htop` è una versione estesa di top scritta in ncurses



Process management: kill/killall

`kill` (util-linux) è un semplice wrapper attorno alla relativa system call. Permette di inviare un segnale ad un processo o gruppo di processi, identificati tramite nome o PID

`kill` (bash) è un comando bash per poter inviare segnali ad un processo identificato tramite PID o JOBSPEC

`killall` permette di inviare un segnale a più processi identificati dal nome

I seguenti comandi sono equivalenti

```
$ /usr/bin/kill -SIGKILL yes
$ kill -9 4929
$ killall -9 yes
```

Process management: pgrep/pkill

`pgrep` permette di ottenere i PID dei processi che soddisfano dei criteri di selezione

`pkill` permette di inviare un segnale ai processi che soddisfano i criteri di selezione

I seguenti comandi sono equivalenti:

```
$ kill $(pgrep telegram)
$ pkill telegram
```

Process management: bash

All'avvio di una shell, verra inizializzata una nuova *sessione*, che può contenere diversi processi controllati dallo stesso terminale

Quando si avvia un programma, bash crea un nuovo processo e lo inserisce in un *process group*. Il processo avrà come file descriptors di *stdout stderr* e *stdin* gli stessi della shell, e avrà come tty di controllo la sessione bash corrente

Il processo verrà eseguito in *foreground*, ossia bash lascia il controllo del terminale al processo fino al termine della sua esecuzione

Process management: bash

Terminata l'esecuzione di un programma, è possibile leggere il suo exit status tramite la variabile `$?`

```
$ ls
file1  file2  file3
$ echo $?
0
```

È possibile interrompere l'esecuzione di un programma in foreground tramite la shortcut Ctrl-C, che invierà un SIGINT al processo

Un processo in foreground può essere sospeso tramite Ctrl-Z, che invierà un segnale SIGSTOP mettendo il processo in stato di stop

Process management: bash

È possibile eseguire un processo in background inserendo un "&" alla fine del comando. La sessione bash tiene traccia di tutti i processi in background, assegnandogli un job ID

- Tramite il comando `jobs` è possibile visualizzare tutti i processi in background lanciati dalla sessione corrente e il loro stato
- `fg` permette di eseguire in foreground un processo precedentemente sospeso o in esecuzione in background. Se non viene specificato un job ID, bash selezionerà l'ultimo processo nella lista
- `bg` continua l'esecuzione in background di processi precedentemente sospesi

Process management: bash

```
$ yes &>/dev/null &
[1] 9758
$ yes &>/dev/null &
[2] 9759
$ jobs
[1]-  Running                 yes &>/dev/null &
[2]+  Running                 yes &>/dev/null &
$ kill -SIGTERM 9758
[1]+  Terminated                    yes &> /dev/null
$ fg
yes &>/dev/null
```

Process management: bash

Il comando `exec` permette di eseguire un programma sostituendo il processo corrente, senza dover creare un nuovo processo. Provate a lanciare il seguente comando:

```
$ exec top
```

Il tool `setsid` permette di eseguire il processo in una nuova sessione invece di ereditare la sessione della shell. Qual'è la differenza tra i seguenti comandi?

```
$ { yes &>/dev/null & } &
$ setsid yes &>/dev/null
```

Thank You!

Domenico Iezzi <domenico.iezzi.201@gmail.com>