Chapitre 6: Processus

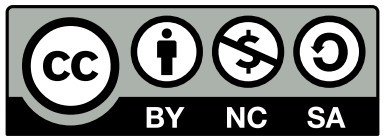
INF1070

Utilisation et administration des systèmes informatiques

Jean Privat & Alexandre Blondin Massé

Université du Québec à Montréal

v213



Plan

Processus (rappel)

- Processus = un programme en cours d'exécution
- `ps`, `top`, etc. pour les voir

Threads

Threads

Synonymes

- Thread
- Fil d'exécution
- Processus léger

Thread et processus

- Un même processus peut avoir plusieurs threads
- Les threads d'un processus ne sont pas isolés
- Mais chaque thread utilise du CPU selon ses besoins

Utilisation: tâches asynchrones, parallélisme

Langages de haut niveau

- Java, C#, Python, etc. ont des threads
- Ne correspondent pas nécessairement aux threads système

Voir les threads

`pstree`

- Affiche les threads par défaut.
- `-T` cacher les threads

`ps`

- `-L` afficher les threads (extra)
- `J` afficher les threads (BSD)

`top`

- `H` bascule thread/processus

Tâches shell (jobs)

Arrière-plan

- « & » passe les commandes en arrière-plan
- & termine et/ou sépare les commandes
- Le shell affiche le numéro de job (entre crochets)
- Et le PID du processus en arrière-plan
- L'invite de commande est à nouveau disponible

```
$ gnome-calculator & xlogo &
[1] 15661
[2] 15666
$ ps
  PID TTY          TIME CMD
 6356 pts/0    00:00:00 bash
15661 pts/0    00:00:00 gnome-calculato
15666 pts/0    00:00:00 xlogo
15669 pts/0    00:00:00 ps
```

Commandes en arrière-plan

Conduites

Des commandes complexes peuvent passer en arrière-plan

```
$ cat /dev/urandom | tr -cd 'ATGC' |
> head -c 10M > adn.txt &
```

Attention

Aux commandes en arrière-plan qui font des affichages

- L'écran contiendra les sorties mélangés
- L'invite du shell peut être noyée

```
$ cat /dev/urandom | tr -cd 'atgc\n' &
```

Ctrl-C ne fonctionne que sur les tâches en avant-plan

- (et pas toujours en fait)

Contrôle des tâches

Le shell offre une gestion des tâches

- Les tâches (*jobs*) sont un concept du **shell**
- Une tâche est un **groupe** de processus
- Chaque commande simple ou conduite est une tâche

Commandes internes du shell

- `jobs` liste les tâches
- `fg` passe une tâche en premier-plan
- `bg` passe une tâche en arrière-plan

Suspension

- ^Z (ctrl + Z) suspend la tâche en premier plan
- Une tâche suspendue ne travaille plus
- On la relance avec `fg` ou `bg`

Contrôle des tâches: exemple

```
$ xeyes -fg blue & xeyes -fg red
[1] 9399
^Z
[2]+  Stoppé                    xeyes -fg red
$ jobs
[1]-  En cours d'exécution      xeyes -fg blue &
[2]+  Stoppé                    xeyes -fg red
$ bg 2
[2]+ xeyes -fg red &
$ # Je ferme le bleu
[1]-  Fini                       xeyes -fg blue
$ jobs
[2]+  En cours d'exécution      xeyes -fg red &
$ fg 2
xeyes -fg red
^C
```

Contrôle des ressources

Ressources

- Plusieurs utilisateurs
- Plusieurs processus
- Un seul ordinateur

$\rightarrow$ Partage des ressources

$\rightarrow$ Contrôle de l'utilisation

Ressources ?

- Mémoire
- CPU (unité centrale)
- Entrées-sorties disque
- Entrées-sorties réseau
- Etc.

Partage et contrôle de ressources

Le système

Par défaut, le système essaye d'allouer les ressources

- De façon efficace
- De façon équitable

L'utilisateur et l'administrateur

- Voient l'état des ressources et leur consommation
- Configurent certaines utilisations de ressources (priorités, limites)

INF3173

- Principes de systèmes d'exploitation

$\rightarrow$ Le détail des politiques, mécanismes, outils et algorithmes

Ressource mémoire

État du système

`free` — mémoire libre et utilisée du système (extra)

```
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi       4,1Gi       7,8Gi       542Mi       3,1Gi        10Gi
Swap:          4Gi          0B       4,0Gi
```

- total: mémoire physique totale
- free: mémoire physique libre
- shared: la mémoire pour les tmpfs (cf. chapitre 3 sur les fichiers)
- buff/cache: mémoire utilisée pour les entrées-sorties
- available: $\approx$ free + buff/cache
- swap: partition d'échange (cf. chapitre 3 sur les fichiers)

Sour linux: `cat /proc/meminfo/` pour plus de détail

Utilisation mémoire des processus

- **Mémoire résidente**: réellement de la RAM utilisée

→ RSS, RES: mémoire physique en ko

→ %MEM: Pourcentage de la mémoire physique du processus (par rapport à la mémoire physique totale)

```
$ ps -eF --sort -rss | head
```

- **Mémoire virtuelle** = mémoire résidente + mémoire *promise*

→ VIRT, VSZ: taille totale de mémoire virtuelle
Code, données, bibliothèques, etc.

```
$ ps -eF --sort -vsz | head
```

- Mémoire promise?

→ Mémoire réservée (allouée) par le processus mais pas encore occupée (surbooking)

→ Mémoire dans la partition d'échange (*swap*)

→ Fichier projeté en mémoire (*mmap*) pas encore chargé.

Mémoire partagée (*shared*)

- Zones mémoire utilisées par plusieurs processus

$\rightarrow$ Zones en lecture seule

$\rightarrow$ Copies temporairement identiques (*copy-on-write*)

- La somme de la mémoire utilisée par chaque processus peut dépasser 100%
- La mémoire partagée n'est libérée que quand le dernier processus qui l'utilise se termine

Tout ça rend la compréhension de l'utilisation de la mémoire encore plus compliquée

Ressource processeur

Charge système (CPU)

`uptime` affiche la durée d'activité et la charge système (extra)

```
$ uptime
 11:30:50 up 23:17, load average: 0,33, 0,58, 0,66
```

Charge

- Nombre moyen de processus dans un état exécutable
- Pour les 1, 5, et 15 dernières minutes.
- Non normalisé sur le nombre de processeurs

« charge = 1 »: en moyenne, un processus travaille

- Monoprocesseur: le processeur est utilisé à 100%
- 4 cœurs: en moyenne 75% des cœurs sont libres

Utilisation CPU des processus

- STIME (et START): date et heure de démarrage

→ date et heure de de naissaince du processus

- TIME: temps total CPU consommé

→ total des petits morceaux de temps où le CPU exécutait le processus

- C (et %CPU): taux d'utilisation du CPU

→ TIME/STIME

```
$ ps -eF --sort -start | head
$ ps -eF --sort -time | head
$ ps -eF --sort -%cpu | head
```

État des processus

- En pratique, la plupart des processus ne travaillent pas
- Ils sont **endormis** (*sleeping*) et attendent un entrée ou un événement:
- Du disque, du réseau, de l'utilisateur, l'expiration d'un délai, etc.

Mesurer le temps

`time` mesure le temps passé pendant l'exécution d'une commande

```
$ time ./travaille
..........
real    0m2,998s
user    0m2,998s
sys 0m0,000s
```

- `real`: temps réel (chronomètre)
- `user`: temps CPU consommé
- `sys`: temps utilisé par le noyau du système d'exploitation

Faire une pause

`sleep` effectue une pause pour une durée déterminée

```
$ time sleep 3

real    0m3,003s
user    0m0,003s
sys 0m0,001s
```

Deux commandes time

Programme GNU autonome

- Ne fonctionne que sur les commandes simples
- Affichage différent et plus d'options

```
$ /usr/bin/time ./travaille
..........
2.99user 0.01system 0:03.01elapsed 100%CPU 21840maxresident
(0major+5196minor)pagefaults 0swap
```

Commande interne Bash

- Fonctionne aussi sur les conduites
- Affichage et options limités

```
$ time echo 1 | sleep 2
real    0m2,004s
$ /usr/bin/time -p echo 1 | sleep 2
real 0.00
```

Priorité d'ordonnancement

`nice` exécute un programme avec une politesse (ou courtoisie) modifiée

- 0: politesse par défaut
- 19: politesse maximale → priorité minimale (Linux)
- -20: politesse minimale → priorité maximale (Linux)

`renice` modifier la politesse d'un processus

```
$ ps -eo nice,pid,user,cmd
```

Politiques habituelles

- Un utilisateur normal peut seulement **augmenter** la politesse de **ses** processus
- root peut **diminuer** la politesse de **tous** les processus
- Les priorités sont **strictes**: un processus pas poli sera toujours prioritaire

Signaux

Signaux

Commande `kill` envoie un signal à un processus

Usage: `kill PID`

- `-l` lister les signaux connus
- `-s` envoyer un signal spécifique
- $-n$ envoyer le signal n°n

Signaux usuels

- `SIGTERM` (15) est envoyé par défaut

$\rightarrow$ Ça demande au processus de se terminer

- `SIGINT` (2) est également envoyé par ^C (Ctrl + C)

$\rightarrow$ Ça demande au processus de se terminer

- `SIGKILL` (9) force la terminaison

$\rightarrow$ Ça termine le processus sans **rien** demander

Commande interne kill

`kill` est aussi souvent une **commande interne** du shell

```
$ type -a fg bg kill
fg est une primitive du shell
bg est une primitive du shell
kill est une primitive du shell
kill est /bin/kill
```

Le `kill` des shell sait envoyer des signaux aux jobs (avec `%`)

```
$ xeyes &
[1] 10446
$ kill %1
```

Qui peut envoyer des signaux à quoi ?

L'utilisateur courant

Peut envoyer à **ses propres** processus

Le super-utilisateur (root)

À **tous** les processus

Le noyau du système d'exploitation

À **tous** les processus

Autres commandes utiles

Rechercher des processus

`pidof` recherche les PID de programmes (LSB)

`pgrep` recherche des processus avec une expression régulière

```
$ pidof /bin/bash
27103 19204
$ pgrep b.sh
19204
27103
```

Envoi de signaux

- `killall` cible un processus par son nom (LSB)
- `pkill` cible un processus avec une expression régulière
- `killall5` cible tous les autres processus du système

Suspendre et reprendre

- `SIGTSTP` est également envoyé par ^Z (Ctrl + Z)

→ Ça demande au processus de se suspendre

- `SIGSTOP` force la suspension

→ Ça suspend un processus de force

- `SIGCONT` reprend un processus suspendu

→ Le processus continue comme si de rien n'était

Détacher du terminal

Par défaut, quitter le shell termine toute les tâches en arrière-plan

```
$ xeyes &
$ exit
```

`nohup` rend insensible une commande aux déconnexions
et redirige les flots standards

- entrée standard = `/dev/null`
- sorties standards = le fichier `nohup.out`

```
$ nohup xeyes &
$ exit
```

La commande interne `disown` détache une tâche existante (Bash)

```
$ xeyes &
$ disown
$ exit
```