# Chapitre 5: Processus

## INF1070

## Utilisation et administration des systèmes informatiques

Jean Privat & Alexandre Blondin Massé

Université du Québec à Montréal

Hiver 2019

# Plan

# Processus

# Processus UNIX

Processus UNIX =
Un programme en cours d'exécution =

- Le programme exécuté (fichier exécutable)
- De l'état et des ressources (mémoire, CPU, etc)
- Un utilisateur (et un groupe)
- Un identifiant = numéro de processus (pid)
- Un processus parent dont il hérite ses caractéristiques
- Un début... et une fin
- D'autres informations utiles à sa bonne gestion

Un processus est un concept du **système d'exploitation**

# Isolation des processus

Les processus sont isolés les uns des autres

- Un processus est autonome et cohérent
- Plein de processus existent en même temps (multitâche)
- Un processus ne peut pas corrompre un autre processus
- Un processus peut collaborer avec d'autres processus

## Exemple de collaboration: tubes

```
$ grep lol /usr/share/dict/french | lolcat
```

2 processus: grep et lolcat

# Lister les processus

Commande `ps`: instantané des processus

```
$ ps
  PID TTY          TIME CMD
 4517 pts/3    00:00:00 bash
24535 pts/3    00:00:00 ps
```

Par défaut `ps` affiche la liste des processus

- De l'utilisateur courant
- Dans le terminal en cours
- Avec peu d'information

```
$ ps | head -n 50 | sort -n | grep ' '
```

# Arborescence des processus

Un nouveau processus est créé par un autre processus

- Un créateur est appelé **processus parent**
- Le premier processus (dit racine) n'a pas de parent
  C'est `init` (ou `systemd`) de PID=1
- Sous UNIX la relation de parenté est préservée

→ On a une arborescence de processus

Commande `pstree` vue arborescente des processus (extra)

```
$ pstree -phT
```

# Options de ps

Trois familles, plein d'options

- Traditionelle (POSIX), avec un tiret « - »
- BSD, sans tiret
- Extensions extra, avec un ou deux tirets « -- »

→ Beaucoup de confusion

```
$ ps -eF
$ ps aux
```

- `-e`, `-A` afficher tous les processus
- `-f` afficher plus de colonnes
- `-F` afficher encore plus de colonnes (extra)
- `a` tous les processus (avec un terminal)
- `ax` tous les processus (même sans terminal)
- `u` afficher des colonnes orientées utilisateur

# Information des processus

```
$ ps -F
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
privat    1435   356  0  3832  2960   7 15:24 pts/0    0:00 ps -F
```

- UID: l'utilisateur du processus
- PID: l'identifiant du processus
- PPID: l'identifiant du processus parent
- C: taux d'utilisation du CPU
- SZ: taille mémoire totale (en pages)
- RSS: taille mémoire résidente (en ko)
- PSR: processeur assigné
- STIME: date et heure de démarrage
- TTY: terminal associé
- TIME: temps total CPU consommé
- CMD: ligne de commande

# Plus d'information des processus

```
$ ps u
USER PID %CPU %MEM VSZ RSS TTY  STAT START TIME COMMAND
jean 252  0.0  0.0 384 353 tty1 R+   10:08 0:00 ps u
```

- USER: l'utilisateur du processus (comme UID)
- PID: l'identifiant du processus
- %CPU: taux d'utilisation du CPU (comme C)
- %MEM: taux d'utilisation de la mémoire (**nouveau**)
- VSZ: taille mémoire totale (comme SZ mais en ko)
- RSS: taille mémoire résidente (en ko)
- TTY: terminal associé
- STAT: état du processus (**nouveau**)
- START: date et heure de démarrage (comme STIME)
- TIME: temps total CPU consommé
- COMMAND: ligne de commande (comme CMD)

# Autres options pratiques de ps

## Filtrer

- `-p` par PIDs
- `-C` par noms de commande (extra)
- `-u` par utilisateurs
- `-x` par l'utilisateur courant (extra)

## Afficher

- `-o` indiquer les colonnes voulues
- `L` lister les colonnes possibles (BSD)
- `--forest` affiche l'arborescence (extra)
- `--sort` trie les lignes (extra)

# 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
```

## Par processus

- VIRT, VSZ, SH: taille totale de mémoire virtuelle
  Code, données, bibliothèques, swap, mémoire partagée, etc.
- RSS, RES: taille de mémoire résidente
  Données réellement en RAM
- SHR: mémoire partagée

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

# 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

# Suivre en temps réel les processus

Commande `top`: processus en temps réel (extra)

- Liste les processus par utilisation processeur
- Interface interactive
- Plein de commandes pour filtrer et trier

## Quelques commandes

- `q` (ou Ctrl+C) quitter
- `h` affiche l'aide
- `P` trier par consommation CPU (défaut)
- `M` trier par consommation mémoire
- `N` trier par PID
- `T` trier par temps CPU total
- `k` terminer un processus

# 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
```

# 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

→ Ça demande au processus de se terminer

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

→ Ça demande au processus de se terminer

- `SIGKILL` (9) force la terminaison

→ 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

# Contrôle des ressources

# 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
```

# 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
```

# Priorité d'ordonnancement

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

- 0: politesse par défaut
- 19: politesse maximale $\rightarrow$ priorité minimale (Linux)
- -20: politesse minimale $\rightarrow$ 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