TP n° 1

Gestion des processus sous GNU/Linux

1. Création d'un processus

On appelle processus un objet dynamique correspondant à l'exécution d'un programme ou d'une commande GNU/Linux. Cet objet regroupe plusieurs informations, en particulier l'état d'avancement de chaque programme, l'ensemble des données qui lui sont propres, ainsi que d'autres informations sur son contexte d'exécution.

Un processus possède des caractéristiques qui permettent au système de l'identifier. Parmi ces caractéristiques, on trouve en particulier :

- l'identifiant du processus (process ID – PID) qui est un nombre entier positif ;
- l'identifiant du processus qui lui a donné naissance, ou processus parent, appelé PPID (parent PID) ;
- l'identifiant de son propriétaire, en général (mais pas toujours) l'utilisateur qui l'a lancé ;
- son répertoire de travail ;
- sa priorité ;
- son temps d'exécution ;
- etc.

L'interpréteur de commandes « ***shell*** » est un programme et il devient un processus lorsqu'il est exécuté. Le programme shell par défaut des systèmes GNU/Linux est appelé « ***bash*** ». Chaque fois qu'une commande est exécutée dans un shell, un nouveau processus est créé à une exception : les commandes internes du shell (*built-in shell*), elles s'exécutent dans le contexte du shell. Il faut utiliser la commande **type** pour vérifier si une commande est une commande interne ou non.

Exemple :

```
etudiant@polytech:~$ type cp ls which type
```

2. Gestion des processus

Il est possible de gérer les processus démarrés dans le système avec la commande **ps**. Pour afficher tous les processus du système, il faut taper :

```
etudiant@polytech:~$ ps -Al
F S  UID   PID  PPID  C PRI NI ADDR  SZ  WCHAN  TTY   TIME       CMD
4 S   0     1     0   0  76  0  -   469    -     ?   00:00:00     init
1 S   0     2     1   0  94 19  -    0   ksofti ?   00:00:00 ksoftirqd/0
1 R 1000 4976  3574  0  76  0  -   7287   -     ?   00:00:01   konsole
```

Le programme **ps** affichera tous les processus actifs, leur PID, PPID, ainsi que d'autres informations.

Options classiques de la commande **ps** :

- -A : afficher tous les processus.
- -l : format long.
- -a : afficher tous les processus, sauf les processus non rattachés à un terminal.
- -u [users] : afficher les processus appartenant à l'utilisateur ayant le nom spécifié.

Exemple :

Pour afficher tous les processus dans la session, taper :

```
etudiant@polytech:~$ ps -l
F S  UID   PID  PPID C PRI NI ADDR  SZ  WCHAN TTY    TIME     CMD
4 S 1000  5448  4981 0  77  0  -   639  wait  pts/1 00:00:00  su
0 S 1000  5450  5448 0  75  0  -   795  wait  pts/1 00:00:00  bash
0 R 1000  7050  5450 0  76  0  -   668   -    pts/1 00:00:00  ps
```

3. Gestion des processus en temps réel

Pour gérer les processus en temps réel, il faut utiliser la commande **top**.

```
etudiant@polytech:~$ top
top - 15:13:13 up  3:53,  2 users,  load average: 1.30, 0.69, 0.53
Tasks: 219 total,   3 running, 216 sleeping,   0 stopped,   0 zombie
Cpu(s): 67.8%us, 20.5%sy,  0.0%ni, 11.4%id,  0.0%wa,  0.3%si,  0.0%st
Mem:   2572660k total,  2553380k used,    19280k free,   120660k buffers
Swap:  2120540k total,     8268k used,  2112272k free,   967508k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1189 root      20   0  312m 167m  66m R   58  6.7  21:53.07 Xorg
16792 etudiant  20   0  393m  22m 3032 S   56  0.9   0:01.72 thunderbird
12071 root      20   0  236m  49m  25m S   28  2.0   2:19.76 synaptic
16791 etudiant  20   0  393m  22m 2992 S   27  0.9   0:00.93 firefox
 1680 etudiant  20   0  338m 103m  64m S    3  4.1   8:20.80 compiz
 2008 etudiant  20   0  112m  19m 4340 S    2  0.8   0:37.28 beam.smp
 1894 etudiant  20   0  137m  30m 5588 S    1  1.2   0:29.50 ubuntuone
 3098 etudiant  20   0  935m 179m  98m S    1  7.1   2:05.15 soffice
16471 etudiant  20   0  329m  32m  16m S    1  1.3   0:01.81 terminal
    1 root      20   0 23828 1908 1288 S    0  0.1   0:00.74 init
```

Avec :

- **%CPU** : Le pourcentage du temps processeur utilisé par le processus.
- **%MEM** : Le pourcentage de la mémoire physique occupée par le processus.

Dès que **top** est lancée, il est possible d'exécuter des commandes interactives, taper :

- **N** : pour classer les processus par PID.
- **A** : pour classer les processus dans l'ordre chronologique (les plus récents en premier).
- **P** : pour classer les processus par rapport à leur utilisation CPU.
- **M** : pour classer les processus par rapport à l'utilisation de la mémoire.
- **k** : pour tuer un processus (le PID sera demandé).

4. Contrôle de l'exécution des processus

La commande **kill** permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas seulement à « tuer » un processus. Il existe de nombreux signaux différents qu'on peut lister avec la commande **kill -l** (vous trouverez la signification dans le manuel : **man 7 signal**). Les signaux les plus courants sont :

- **SIGSTOP** (19) pour arrêter un processus.
- **SIGCONT** (18) pour continuer un processus arrêté.
- **SIGTERM** (15) pour signifier au processus qu'il doit se terminer.
- **SIGKILL** (9) pour tuer un processus.

La syntaxe d'envoi d'un signal est : **kill -signal pid** où **signal** est un numéro ou un nom de signal (le signal par défaut est **SIGTERM**).

Exemples :

```
etudiant@polytech:~$ kill -SIGKILL [PID]
etudiant@polytech:~$ kill -15 [PID]
```

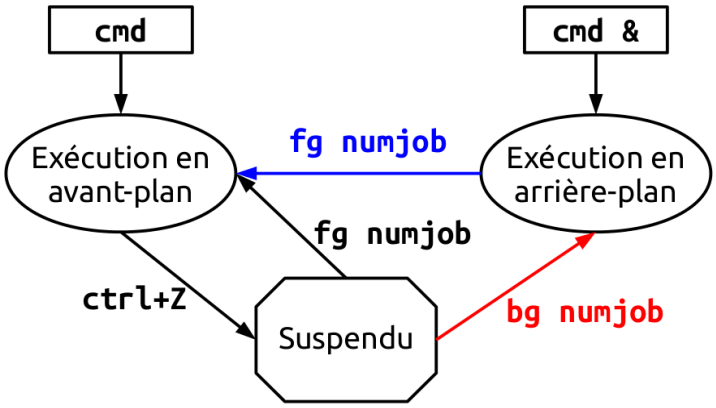
5. Contrôle des tâches

Dans un processus ***bash***, il est possible de démarrer plusieurs processus appelés aussi ***jobs***. Par défaut, un processus est démarré en avant-plan et il est le seul à recevoir les données de l'entrée standard (le clavier). Il faut utiliser **Ctrl+Z** pour le suspendre. Pour démarrer un processus en arrière-plan, il faut utiliser le signe **&**.

Exemples :

```
etudiant@polytech:~$ gedit document.txt &
etudiant@polytech:~$ xeyes &
```

Les commandes pour manipuler les jobs sont les suivantes :



- Pour démarrer un processus en arrière-plan, il faut utiliser le caractère **&** :

```
etudiant@polytech:~$ xeyes &
[1] 12098
etudiant@polytech:~$
```

- Pour lister tous les jobs actifs :

```
etudiant@polytech:~$ jobs
[1]+ Running      xeyes &
etudiant@polytech:~$
```

- Ramener un job en avant-plan :

```
etudiant@polytech:~$ fg %1
xeyes
```

- Suspendre un job, taper **Ctrl+Z**, le job passe dans un état arrêté :

```
[1]+ Stopped xeyes
etudiant@polytech:~$
```

- Continuer l'exécution d'un job tournant en arrière-plan :

```
etudiant@polytech:~$ jobs
[1]+  Stopped      xeyes
etudiant@polytech:~$ bg %1
[1]+ xeyes &
etudiant@polytech:~$
```

- Tuer un job :

```
etudiant@polytech:~$ kill %1
```

Lorsque la session *bash* est terminée, tous les processus démarrés depuis ce shell reçoivent le signal **SIGHUP**, ce qui, par défaut, les stoppe. Dans le dernier exemple, le fait de tuer le processus *bash* tuera aussi le processus xeyes. Pour éviter qu'un processus ne s'arrête lorsque son parent se termine, le programme peut être lancé avec la commande **nohup** :

```
etudiant@polytech:~$ nohup xeyes
```

6. Questions

1. Quelle commande vous permet de vérifier facilement quel utilisateur consomme la plus grande partie des ressources processeur du système ?
2. Est-ce que tous les processus ont un processus parent ?
3. Qu'arrive-t-il si un processus enfant perd son parent ?
4. Quel est le nom d'un programme ou d'une application active ?
5. Quelle commande permet d'envoyer un signal à un processus ?
6. De quel élément avez-vous besoin pour envoyer un signal à un processus ?
7. Quel signal reçoivent tous les processus lorsque la session *bash* à partir de laquelle ils ont été lancés se termine ?

7. Exercice

1. Lancer deux terminaux (tty1 et tty2)
2. Exécuter dans le tty1 la ligne de commande suivante :

```
etudiant@polytech$ (while true; do echo -n A >> /tmp/log; sleep 1; done)
```

Que remarquez-vous ? Analyser cette ligne de commande en expliquant chaque partie. Essayer d'ouvrir le fichier **/tmp/log** pour vérifier son contenu.

3. Dans le terminal tty2, utiliser la commande (**tail -f**) pour afficher en temps réel le contenu du fichier **/tmp/log**
4. Dans le tty1, interrompre le processus en cours d'exécution en utilisant **Ctrl+Z**.
5. Quel est le numéro de travail associé au processus interrompu ?
6. Vérifier dans tty2 que le remplissage du fichier log est arrêté.
7. Dans le tty1, lancer à nouveau le processus arrêté mais en arrière plan.
8. Vérifier que le travail déjà arrêté est en cours d'exécution. Vérifier dans tty2, l'état de fichier **/tmp/log**
9. Dans tty1 et lancer les deux commandes suivantes :

```
etudiant@polytech$ (while true; do echo -n B >> /tmp/log; sleep 1; done) &
etudiant@polytech$ ^B ^C
```

Expliquer le rôle de ces deux commandes.

10. Dans tty2, vérifier que les trois processus sont en cours d'exécution et consulter le fichier **/tmp/log**
11. Dans tty1, utiliser ensuite **kill** pour envoyer un **SIGSTOP** au numéro de travail du premier processus exécuté pour le suspendre. Vérifier avec la commande **jobs**
12. Relancer le travail du premier processus à l'aide de **kill** et le signal **SIGCONT**.
13. Utiliser **SIGTERM** pour terminer les travaux [2] et [3]. Vérifier avec jobs qu'ils sont terminés.
14. Vérifier si tous les processus ont été terminés, sinon terminer ceux qui restent.