Gestion des processus

Les systèmes d'exploitation de type Unix sont multitâches et multi-utilisateurs. Cela signifie qu'il est possible d'exécuter simultanément plusieurs programmes qui appartiennent potentiellement à différents utilisateurs. Sous Unix, l'unité d'exécution d'un programme est appelée un **processus**. Lorsque vous exécutez un programme C que vous avez compilé depuis la ligne de commande, le shell lance un nouveau **processus**. Chaque processus est identifié par le système d'exploitation via son **pid** ou **process identifier**. Ce **pid** est alloué par le système d'exploitation au moment de la création du processus. À tout instant, le système d'exploitation maintient une **table des processus** qui contient la liste de tous les processus qui sont en cours d'exécution. Comme nous aurons l'occasion de nous en rendre compte plus tard, cette table contient énormément d'informations qui sont utiles au système. À ce stade, l'information importante qui se trouve dans la table des processus est le **pid** de chaque processus et l'utilisateur qui possède le processus. La commande **ps(1)** permet de consulter de façon détaillée la table des processus sur un système Unix. Voici un exemple d'utilisation de cette commande sur un système Linux.

```
$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
obo      16272  0.0  0.0 110464  1884 pts/1    Ss   11:35   0:00 -bash
obo      16353  0.0  0.0 110184  1136 pts/1    R+   11:43   0:00 ps u
```

Dans cet exemple, l'utilisateur `obo` possède actuellement deux processus. Le premier est l'interpréteur de commande **bash(1)** et le second le processus **ps(1)**. L'interpréteur de commande a `16272` comme **pid** tandis que le **pid** de **ps(1)** est 16353.

ps(1) n'est pas la seule commande permettant de consulter la table des processus. Parmi les autres commandes utiles, on peut mentionner **top(1)** qui permet de visualiser les processus qui s'exécutent actuellement et le temps CPU qu'ils consomment ou **pstree(1)** qui présente les processus sous la forme d'un arbre. Sous Linux, le répertoire `/proc`, qui est documenté dans **proc(5)** contient de nombreux pseudos fichiers avec énormément d'informations relatives aux processus qui sont en cours d'exécution. Parcourir le répertoire `/proc` et visualiser avec `less(1)` les fichiers qui s'y trouvent est une autre façon de consulter la table des processus sous Linux.

Pour comprendre le fonctionnement des processus, il est intéressant d'expérimenter avec le processus ci-dessous. Celui-ci utilise l'appel système **getpid(2)** pour récupérer son **pid**, l'affiche et utilise la fonction **sleep(3)** de la librairie pour se mettre en veille pendant trente secondes avant de se terminer.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {

  unsigned int sec=30;
  int pid=(int) getpid();

  printf("Processus : %d\n",pid);
  printf("[pid=%d] Sleep : %d secondes\n",pid, sec);
  sec=sleep(sec);
  if(sec==0) {
    printf("[pid=%d] Fin du processus\n",pid );
    return(EXIT_SUCCESS);
  }
  else {
    printf("[pid=%d] Interrompu alors qu'il restait %d secondes\n",pid,sec);
    return(EXIT_FAILURE);
  }
}
```

Ce programme peut être compilé avec **gcc(1)** pour produire un exécutable.

```
$ ls -l getpid*
-rw-r--r--  1 obo  obo   608 10 fév 12:11 getpid.c
$ gcc -Wall -o getpid getpid.c
$ ls -l getpid*
-rwxr-xr-x  1 obo  obo  8800 10 fév 12:12 getpid
-rw-r--r--  1 obo  obo   608 10 fév 12:11 getpid.c
```

Cet exemple utilise la commande **ls(1)** pour lister le contenu d'un répertoire. L'argument `-l` permet de d'obtenir pour chaque fichier son nom, sa date de modification, sa taille, l'utilisateur et le groupe auquel il appartient ainsi que ses permissions. Sous Unix, les permissions associées à un fichier sont divisées en trois blocs. Le premier bloc correspond aux permissions qui sont applicables à l'utilisateur qui possède le fichier. Pour l'exécutable `getpid`, les permissions du propriétaire sont `rwx`. Elles indiquent que le propriétaire peut lire le fichier (permission `r`), l'écrire ou l'effacer (permission `w`) et l'exécuter (permission `x`). Sous Unix, seuls les fichiers qui possèdent la permission à l'exécution peuvent être lancés depuis l'interpréteur. Ces permissions peuvent être modifiées en utilisant la commande **chmod(1)**. Les deux autres blocs de permissions sont relatifs aux membres du même groupe que le propriétaire et à un utilisateur quelconque. Nous y reviendrons plus en détail lorsque nous abordons les systèmes de fichiers. En pratique, il est important de savoir qu'un fichier shell ou un fichier compilé qui n'a pas le bit de permission `x` ne peut pas être exécuté par le système. Ceci est illustré par l'exemple ci-dessous.

```
$ chmod -x getpid
$ ls -l getpid*
```

```
-rw-r--r--  1 obo  obo  8800 10 fév 12:12 getpid
-rwxr-xr-x  1 obo  obo  8800 10 fév 12:11 getpid.o
$ ./getpid
-bash: ./getpid: Permission denied
$ chmod +x getpid
$ ./getpid
Processus : 11147
```

L'interpréteur de commande **bash(1)** permet lancer plusieurs processus en tâche de fond. Cela se fait en suffixant la commande avec `&`. Des détails complémentaires sont disponibles dans la section `JOB CONTROL` du manuel de **bash(1)**. Lorsqu'un processus est lancé en tâche de fond, il est détaché et n'a plus accès à l'entrée standard. Par contre, il continue à pouvoir écrire sur la sortie standard et la sortie d'erreur standard. L'exemple ci-dessous illustre l'exécution de deux instances de `getpid`.

```
$ ./getpid &
[1] 10975
$ Processus : 10975
[pid=10975] Sleep : 30 secondes
$ ./getpid &
[2] 10976
$ Processus : 10976
[pid=10976] Sleep : 30 secondes
ps u
USER   PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
obo   8361   0,0  0,0  2435548    208 s003  S+    9:24     0:00.14 -bash
obo  10975   0,0  0,0  2434832    340 s000  S    12:05     0:00.00 ./getpid
obo  10976   0,0  0,0  2434832    340 s000  S    12:05     0:00.00 ./getpid
[pid=10975] Fin du processus
[pid=10976] Fin du processus
[1]-  Done                    ./getpid
[2]+  Done                    ./getpid
```

Ces deux instances partagent la même sortie standard. En pratique, lorsque l'on lance un processus en tâche de fond, il est préférable de rediriger sa sortie et son erreur standard. Lorsque l'on développe de premiers programmes en C, il arrive que celui-ci se lance dans une boucle infinie. Deux techniques sont possibles pour interrompre un tel processus qui consomme inutilement les ressources de la machine et peut dans certains cas la surcharger fortement.

Si le programme a été lancé depuis un shell, il suffit généralement de taper sur *Ctrl-C* pour interrompre son exécution, comme dans l'exemple ci-dessous.

```
$ ./getpid
Processus : 11281
[pid=11281] Sleep : 30 secondes
^C
```

Parfois cependant *Ctrl-C* n'est pas suffisant. C'est le cas notamment lorsqu'un processus a été lancé en tâche de fond. Dans ce cas, la meilleure technique est d'utiliser **ps(1)** pour trouver l'identifiant du processus et l'interrompre via la commande **kill(1)**. Cette commande permet d'envoyer un **signal** au processus. Nous verrons plus tard le fonctionnement des signaux sous Unix. À ce stade, le signal permettant de terminer avec certitude un processus est le signal `KILL`. C'est celui qui est utilisé dans l'exemple ci-dessous.

```
$ ./getpid &
[1] 11285
$ Processus : 11285
[pid=11285] Sleep : 30 secondes
ps
PID TTY           TIME CMD
384 ttys000      0:00.32 -bash
11285 ttys000    0:00.00 ./getpid
$ kill -KILL 11285
$ ps
 PID TTY           TIME CMD
 384 ttys000    0:00.33 -bash
 [1]+  Terminated              ./getpid
```