TP 1

Gestion des processus

Chaque programme en cours d'exécution est un “processus”. Il en existe plusieurs en exécution, ceux lancés par vous-même, et ceux lancés par d'autres utilisateurs ou par l'utilisateur « root ». Vous avez le contrôle sur tous les processus que vous démarrez. En outre, l'utilisateur « root » a le contrôle sur tous les processus du système. Les processus peuvent être contrôlés et analysés par différents programmes existants, ainsi que par des commandes du terminal (ou le *shell).* Dans les TPs consacrés aux processus, nous allons manipuler un certain nombre de commandes shell permettant de contrôler les processus Linux (partie facultative). Dans la deuxième partie, nous approfondirons notre étude des processus par le développement de nos propres programmes en langage C, permettant la création, la manipulation, l'analyse et l'arrêt des processus (partie obligatoire).

1 Contrôle des processus

1.1 Travaux en arrière-plan (backgrounding)

Chaque programme lancé à partir de la ligne de commande est démarré en *premier plan.* Cela vous permet de voir tous les résultats du programme et d’interagir avec. Cependant, il y a plusieurs occasions où vous souhaitez que le programme fonctionne sans perdre le contrôle de votre terminal. C'est ce qu'on appelle l'exécution du programme en *tâche de fond* ou en *arrière-plan*, et il y a plusieurs façons de le faire.

La première façon est d'ajouter une esperluette (&) à la fin de la ligne de commande lorsque vous démarrez un programme.

MANIPULATION 1

1. Allez dans le dossier `TP1` et compilez le programme `chrono.c`

```
$ gcc chrono.c -o chrono
```

2. Dans deux terminaux séparés, lancez le programme `chrono` avec et sans le `&`.

a. Dans terminal 1 `$ ./chrono`

b. Dans terminal 2 `$ ./chrono &`

3. Pour chaque manipulation, lors de l'exécution du programme,

a. tapez la commande `ls` qui affiche le contenu du dossier courant. Quelle constatation peut-on faire ?

b. cliquez sur `Ctrl + c` qui permet de terminer le processus en cours. Quelle constatation peut-on faire ?

A noter que le programme `chrono` se termine tout seul après 20 itérations. S'il a été lancé en arrière-plan, à la fin de son exécution un message comme ci-dessous sera affiché :

```
[1]+  Done    ./chrono
```

L'autre façon de mettre un processus en tâche de fond est de le faire pendant qu'il est en marche. En appuyant sur `Ctrl + z` , on suspend le processus en cours d'exécution et on le met en pause. Il est donc possible de mettre un processus en arrière-plan en tapant :

```
$ bg
```

MANIPULATION 2

4. Lancez le programme `chrono` dans un terminal (sans le &).

```
$ ./chrono
```

5. Testez l'utilisation de la commande `ls`

6. Mettez le processus en arrière-plan et testez à nouveau avec la commande `ls`

1.2 Travaux en premier plan (Foregrounding)

Si vous avez besoin d'interagir avec un processus en arrière-plan, il faut le ramener au premier plan. Si vous avez seulement un processus en tâche de fond, vous pouvez le ramener en tapant simplement :

```
$ fg
```

Cependant, il est possible d'avoir plusieurs processus à la fois en arrière-plan. Dans ce cas, vous aurez besoin de savoir quel processus vous souhaitez ramener en premier plan. En tapant juste la commande `fg`, cela va ramener au premier plan le dernier processus mis en arrière-plan. Heureusement, le shell fournit une commande permettant de lister tous les processus en arrière-plan. C'est la commande `jobs` :

```
$ jobs
[1]   Stopped      vi
[2]-  Stopped      man ps
[3]+  Stopped      ./chrono
```

Comme vous pouvez le voir, tous les processus en arrière-plan dans l'exemple sont arrêtés. Les nombres entre parenthèses sont une sorte d'identité. Le processus avec un signe plus + à côté de son identificateur est le processus qui sera au premier plan si vous tapez `fg`.
Si vous voulez par exemple `man ps` en premier plan, vous devez taper :

```
$ fg 2
```

Dans ce cas-là, le processus `man ps` - affichant l'aide de la commande `ps` - revient à la console.

MANIPULATION 3

7. Affichez les pages d'aide des commandes suivantes dans un seul terminal et en même temps : `ps` , `kill` , `top` et `exec` .
Exemple : `$ man ps`
8. Basculez d'une page à l'autre, tout en lisant la description de chacune de ces commandes.

1.3 ps

La commande `ps` permet d'obtenir la liste des programmes en cours d'exécution sur votre terminal. Cela inclut les processus en premier plan et en arrière-plan qui vous appartiennent.

```
$ ps
  PID TTY          TIME CMD
 3168 pts/1    00:00:00 bash
 3302 pts/1    00:00:00 ps
```

La colonne PID est l'identificateur unique du processus. La colonne TTY indique sur quel terminal le processus est exécuté. Si le processus n'est attaché à aucun terminal (un processus *démon* par exemple*),* on aura ?. La colonne TIME indique sur combien de temps CPU le processus a fonctionné. Finalement, la colonne CMD montre le nom du programme.

Vous pouvez obtenir une liste complète des processus en cours d'exécution sur votre système en utilisant la bonne combinaison d'options.

MANIPULATION 4

9. Effectuez des recherches pour trouver la définition de la notion de processus *démon*. Donnez cette définition.
10. Ouvrez deux terminaux différents. Sur l'un d'eux essayez la séquence de commandes suivantes. Voir l'aide de la commande `ps` afin de comprendre le résultat de chaque option.

a. `$ ps -e` quelle est la différence avec un `ps` seul ?
b. `$ ps -f` que représentent les colonnes `UID`, `PPID` et `STIME` ?
c. Combiner les deux options précédentes et retrouver les ascendants du processus `ps`.
d. Quel est le processus père de tous les autres processus ?
e. `$ ps U root` que fait cette commande ? Il est aussi possible de mettre son nom d'utilisateur à la place de « root »

1.4 kill

Si un processus se comporte mal et vous avez besoin de l'arrêter, vous pouvez utiliser la commande `kill`. Afin de tuer un processus, vous aurez besoin de connaître son PID ou son nom (pour cela, vous pouvez utiliser la commande `ps` vue dans la section précédente). Par exemple :

```
$ kill 3128
```

Notez que vous devez être le propriétaire du processus afin de le tuer. L'utilisateur « root » peut tuer n'importe quel processus dans le système.

Une autre variante de `kill` est `killall`. Cette commande permet d'arrêter tout un ensemble de processus ayant un nom particulier.

Parfois, un `kill` ne fait pas le travail. Vous aurez besoin d'utiliser une forme plus puissante. Si un processus ne répond pas à votre volonté de l'arrêter, vous pouvez procéder comme suit :

```
$ kill -9 3128
```

Un `kill` régulier envoie un signal SIGTERM au processus cible, lui demandant de finir ce qu'il est en train de faire, nettoyer son environnement et s'arrêter. L'option -9 envoie un signal SIGKILL qui le tue directement sans lancer les procédures d'arrêt régulières, ce qui peut engendrer des données corrompues. L'option suivante affichera la liste des différents signaux possibles. Voir l'aide de la commande `kill` pour en savoir plus.

```
$ kill -l
```

1.5 top

Il existe une commande qui vous permet de voir les mises à jour instantanées de la liste des processus en cours d'exécution.

```
$ top
```

MANIPULATION 5

11. Consultez la page d'aide de la commande `top`, et trouvez la bonne option pour afficher seulement les processus qui vous appartiennent.

2 Création de processus (fonction fork)

On va tout d'abord utiliser les fonctions UNIX suivantes :

- **`fork()`** :

Cette fonction crée un processus. La valeur de retour *n* de cette fonction indique :

1. $n > 0$

On est dans le processus père, et n indique l'identifiant du processus fils.

2. $n = 0$

On est dans le processus fils,

3. $n = -1$

`fork()` a échoué, on n'a pas pu créer de processus.

- **`getpid()`** :

Cette fonction renvoie le numéro du processus courant.

- **`getppid()`** :

Cette fonction renvoie le numéro du processus père du processus courant.

EXERCICE 1

1. Après compilation de `exo1.c`, qui se trouve en annexe, exécutez le programme `exo1` plusieurs fois.

```
$ gcc exo1.c -o exo1
$ ./exo1
```

Pourquoi cinq lignes sont-elles affichées alors qu'il n'y a que trois `printf` dans le programme?

2. Ajoutez maintenant la ligne suivante après l'appel à `fork()`:

```
if (valeur == 0) sleep (4);
```

Que se passe-t-il ? Pourquoi ?

EXERCICE 2

3. Compilez le programme `exo2.c`, donné en annexe.

Relevez les numéros des processus et numérotez l'ordre d'exécution des instructions `printf()` de façon à retrouver l'ordre d'exécution des processus.

Remarque:
Si on relève un numéro de processus égal à 1, il s'agit du processus `init`, père de tous les processus. `init` adopte les processus orphelins, c'est-à-dire qu'un processus dont le père s'est terminé devient fils de 1, et `getppid()` renvoie 1.

Annexe

exo1.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
  int valeur;
  printf("printf-0 Avant fork: ici processus numero %d\n",(int)getpid());
  valeur = fork();
  printf("printf-1 Valeur retournee par la fonction fork: %d\n", (int)valeur);
  printf("printf-2 Je suis le processus numero %d\n", (int)getpid());
  return 0;
}
```

exo2.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
  int retour_fork1;
  int retour_fork2;
  printf("PID \t PPID \t INSTRUCTION \t RETOUR-FORK\n");
  printf("%d \t %d \t print 1 \t \n", (int)getpid(), (int)getppid());
  retour_fork1 = fork();
  printf("%d \t %d \t print 2 \t %d \n", (int)getpid(), (int)getppid(), retour_fork1);
  retour_fork2 = fork();
  printf("%d \t %d \t print 3 \t %d \n", (int)getpid(), (int)getppid(), retour_fork2);
  return 0;
}
```