TP Linux
Introduction aux processus Linux

Les zones en bleu sont des manipulations à faire.

Généralités :

Un processus correspond à un programme en cours d'exécution par le système d'exploitation, associé à un environnement nécessaire à son bon fonctionnement.

Un système multitâches peut gérer simultanément plusieurs processus.
Mais à un instant donné, un processeur ne s'occupe que d'un seul processus.
Un Ordonnanceur (scheduler) répartit le temps CPU entre les différents processus. Ce multiplexage temporel du traitement donne à l'utilisateur l'impression que tous les processus s'exécutent en même temps.

1. Environnement de travail :

Vous devez faire ce TP sur un shell Linux. Si la salle ne dispose pas d'accès Linux, vous pouvez utiliser le site https://codeanywhere.com/signup vous inscrire, et créer un container C++ (vous disposerez d'un accès shell).

2. Identifier les processus en cours :

Le système d'exploitation maintient à jour une table des processus en cours.

La commande shell *ps* permet de les visualiser avec plus ou moins de détails suivant l'option utilisée (ps –ef ; ps –af).

Chaque processus est identifié par un numéro : le PID (Process IDentifier).
Le premier processus (init) pour le numéro 1.

Parmi les autres caractéristiques, on peut citer :

- Le UID et le GID de l'utilisateur qui est à l'origine de son exécution
- Le UID et le GID effectif du processus (en cas de changement par *setuid* ou *setgid*)
- Le PID du processus père (PPID) : C'est le processus qui est à l'origine d'un autre processus.
- Le temps CPU et l'espace mémoire utilisé par le processus
- L'état du processus et son niveau de priorité (en fonction de l'Ordonnanceur), …

1. Taper la commande : ps
   et la commande ps -ef | more

   Cette commande permet de visualiser la liste des processus en cours d'exécution (au moment où on tape la commande!). Sans les options, seuls les processus de l'utilisateur sont affichés.
   Le « | more » permet d'avoir l'affichage page par page.

   Un certain nombre d'informations sont ainsi affichées. On peut voir, pour un processus : son PID (Process Identifier), son PPID (Parent PID), son UID (User Identifier), sa fenêtre de sortie standard (TTY) s'il en a une, et bien sur le nom de l'exécutable (COMMAND).

   Notez que la commande ps comporte un nombre impressionnant de combinaisons d'options qui permettent de sélectionner précisément ce que l'on veut afficher. Il existe également deux formats de commande : les options sont précédées d'un tiret, et les options indiquées sans le tiret. Suivant le cas, le résultat n'est pas le même. Avec l'habitude, on retient généralement une ou deux commandes ps selon son goût !

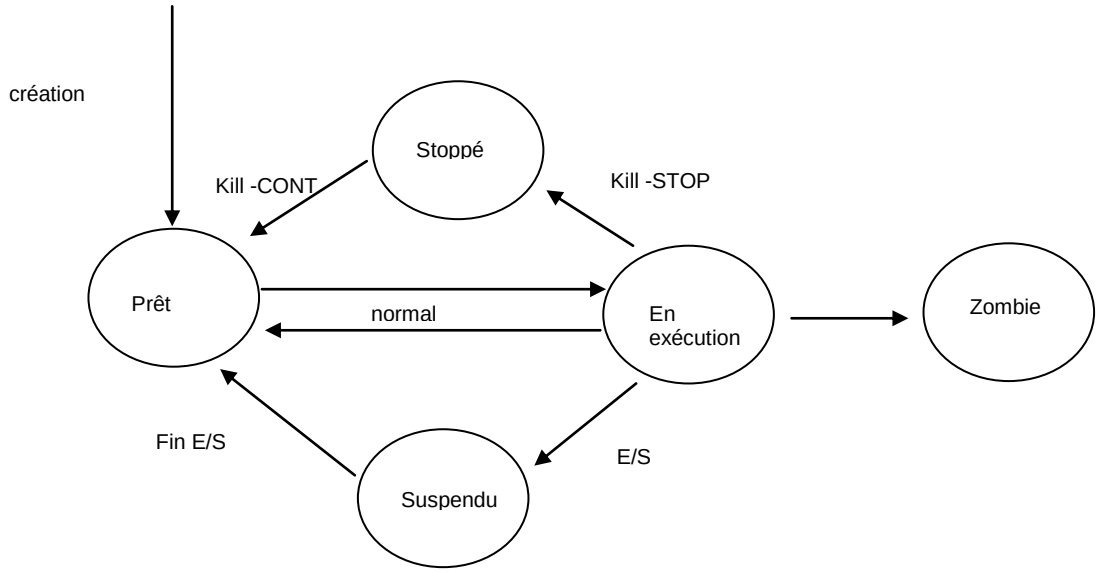
2. Testez les commandes ps afu et ps al
   Ces commandes affichent vos processus personnels.
   Observez les deux façons de représenter la filiation des processus.
   POUR LA SUITE DU TP : vous utiliserez la commande « ps afu »

3. Pour créer une nouvelle filiation, utilisez la commande *su* et observez la filiation des processus.

3. Intervenir sur les processus

3.1. Etats d'un processus :

L'Ordonnanceur ne s'occupant que d'un processus à la fois, un processus peut être :
R (running) : En cours d'exécution par le processeur.
S (sleeping) : Prêt mais un autre processus est en cours d'exécution.
D (defunct) : En attente d'une ressource (ex : la fin d'une e/s).
T (sToped) : Processus stoppé par une intervention extérieure (kill –STOP)
Z (zombie) : Processus terminé mais encore référencé (le père n'attend pas son *exit_signal*).



3.2. Changer l'état d'un processus

Un processus est réceptif à des signaux envoyés par le système d'exploitation (forme d'interruption logicielle). La technique de programmation utilisant les signaux sera décrite dans un autre chapitre.
Par le *shell*, la commande *kill* permet d'envoyer un signal à un processus :

`kill –SIGSTOP <n°PID>`	Stoppe l'exécution du processus par l'Ordonnanceur
`kill –SIGCONT <n°PID>`	Reprend l'exécution du processus
`kill -9 <n°PID>`	Force le processus à se terminer (abandon)
…	

Pour un processus en avant-plan (c'est-à-dire qui monopolise la console à partir de laquelle il a été lancé), on peut interrompre le processus avec <CTRL C> (équivaut à un kill -2 (ou SIGINT)) ;

La séquence <CTRL Z> équivaut à un kill –SIGSTOP : Le processus n'est pas supprimé, il est juste mis en pause.

La commande `bg 1` envoi un kill -SIGCONT au processus que l'on vient de suspendre par <CTRL Z>. Le processus continuera son exécution mais en arrière plan (background).

Si on désire ramener l'exécution de ce processus en avant plan (foreground), utiliser la commande `fg 1`. Le chiffre 1 représente le dernier processus manipulé. Si on en manipule plusieurs, le numéro change (2,3,4, …).

3.2.1. Créer un script de test

Avec un éditeur (exemple : nano), créez un script nommé **tempo.bash** contenant ceci :

```
declare -i cpt
cpt=0

echo "Bonjour je suis le processus $$"

while [ $cpt -ne 200 ]
do
        sleep 1  # pause 1 sec
        cpt=$cpt+1
        echo -ne "$cpt "
done
```

Rendez ce script exécutable en tapant la commande : `chmod +x tempo.bash`
Et lancez l'exécution du script en tapant : `./tempo.bash`

Nous allons observer le comportement de ce script qui s'exécute pendant 200 secondes :

Sur une 2e fenêtre shell, utilisez les commandes « ps » pour repérer le processus correspondant au script et relevez si filiation et son « état ».

3.2.1. Commande kill

La commande ***kill***[1] envoie un signal à un processus en précisant le PID du processus concerné.
`kill -n°du_signal PID_du_processus`

La commande « `kill -l` » affiche la liste des signaux disponibles (lettre L en minuscule)

Utilisez les commandes « kill » vue ci-dessus pour modifier son état : SIGSTOP SIGCONT SIGKILL
A chaque fois, vérifiez l'état affiché avec la commande « ps ».

3.2.2. Arrière plan / Avant plan

Modifiez le script : Supprimez les lignes « echo » pour rendre le script silencieux.

Relancez le script en le plaçant en arrière plan : `./tempo.bash &`

NB : Le shell vous renvoi un numéro de « travail » (job).

Observez l'état du processus ;
Il est possible de « ramener » le processus en premier plan :
`fg 1` où 1 représente le numéro de *job*
On peut obtenir la liste des « jobs » avec la commande : `job`
Vérifiez à nouveau l'état du processus ;

[1] kill contrairement à sa traduction, ne signifie pas uniquement « tuer » sous linux

4. Vue d'ensemble des processus

La commande ***top*** affiche les informations « en temps réel » sur les processus, en ajoutant d'autres informations système. Il existe un outil graphique similaire sous Gnome : Gnome System Monitor.

1. Utilisez ***top*** et observez les différents paramètres du système : %CPU, Load average, STAT, …
   Combien de processus sont simultanément en état R (Running) ?

Surcharger le système :
Pour surcharger un peu le système nous allons lancer un script qui prend beaucoup de temps processeur.

2. Ecrire le script suivant (boucle folle) :

```
a=3
while [ $a -eq 3 ]
do
      a=3
done
```

   Observer l'évolution de la charge du(des) CPU(s). Faites vérifier votre travail.

5. (snir) Les signaux vu du langage C/C++

5.1. Modifier le comportement d'un signal

Créez le source suivant et compilez-le :

```
#include<iostream> // C++

#include<unistd.h> // pour la function usleep
#include<signal.h> // Pour gerer les signaux

using namespace std;

// Fonction appelee par le signal
void mon_code(int sig)
{
    cout << "Arret interdit (Signal : "<< sig << ") !!!" << endl;
}

// Programme principal :
int main ()
{
        // Modification du comportement du signal SIGINT:
        struct sigaction toto; // Structure pour les signaux
        sigaction(SIGINT, NULL, &toto); //on remplit la structure
        toto.sa_handler=mon_code; // On modifie le handler
        sigaction(SIGINT, &toto, NULL); // On applique le changement


        // Programme principal (boucle lente)
        int cpt = 0;

        while ( cpt != 200 )
        {
                usleep(1000000);
                cpt++;
        }
        return 0;
}
```

Exécutez le programme et testez la commande CTRL C.

Quel est le seul moyen de stopper le processus ?

5.2. Envoyer un signal

Créez le source suivant et compilez-le :

```
#include<iostream>
#include<cstdlib> // commande system
#include<signal.h> // commande kill

using namespace std;

int main ()
{
        int pid;

        system("clear");
        cout << "Liste des processus :" << endl;
        system("ps afu");
        cout << endl << "Quel PID supprimer ? ";

        cin >> pid;
        kill(pid, SIGKILL);
        return 0;
}
```

Testez le programme (Etablir une méthode de test)