TP N° 1 : gestion des processus

Les objectifs spécifiques de TP système

Au terme de ces travaux pratiques basés sur le Système Linux, l'étudiant doit être à la hauteur des compétences opérationnelles suivantes :

- exécuter les principales commandes de gestion de processus sur linux (ps, kill,jobs)
- Utiliser le langage C pour créer et manipuler les processus à travers les appels systèmes,
- créer et manipuler plusieurs processus en se servant de l'appel système fork(),
- utiliser l'appel système wait() pour gérer l'ordre d'exécution de plusieurs processus,

Partie N°1 : Visualiser les processus

1. Commande ps

On peut visualiser les processus qui tournent sur une machine avec la commande : **ps** (options), les options les plus intéressantes sont **-e** (affichage de tous les processus) et **-f** (affichage détaillée).

Taper la commande ps -ef
Taper la commande *ps -aux |more*

```
UID     PID  PPID  C STIME      TTY      TIME    COMMAND
root    1    0     0 Dec 6      ?        1:02    init
...
jean    319 300    0 10:30:30   ?        0:02    /usr/dt/bin/dtsession
olivier 321 319    0 10:30:34   ttyp1    0:02    csh
olivier 324 321    0 10:32:12   ttyp1    0:00    ps -ef
```

La signification des différentes colonnes est la suivante:

- **UID** nom de l'utilisateur qui a lancé le process
- **PID** correspond au numéro du process
- **PPID** correspond au numéro du process parent
- **C** au facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
- **STIME** correspond à l'heure de lancement du processus
- **TTY** correspond au nom du terminal
- **TIME** correspond à la durée de traitement du processus
- **COMMAND** correspond au nom du processus.

Cette commande permet de visualiser la liste des processus en cours d'exécution. 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), son **état** (STAT) et sa fenêtre de sa sortie standard (**TTY**) s'il en a une.

2. Commande top

Lancez la commande *top*.

Vous avez un affichage des processus de haut en bas selon un ordre préétabli.

Voila les informations affichées par la commande top

PID	Le PID du processus
USER	L'utilisateur qui exécute ce processus
PR	La priorité de la tâche
NI	Le nice de la tâche
VIRT	Taille virtuelle d'un processus, c'est la somme de la mémoire qu'il utilise réelement en mémoire qu'en mémoire vive), toute mémoire : vidéo (serveur X), bibliotèques, sémaphores ... C'est la quantité de mémoire dont le processus a accès immédiatement
RES	Quantité de mémoire physique occupée par le processus, c'est la taille de la colonne MEM
SHR	Indique quelle quantité de la colonne VIRT réellement partagée
S	Statut du processus. Les valeurs possibles sont : **S** (sleeping), **D** (uninterruptible sleep), **R** (running), **Z** (zombie), **ouT** (stopped or traced), peut être précédé par **<** (negative nice value), **N** (positive nice value), ou **W** (swapped out).
%CPU	Charge CPU
%MEM	Charge mémoire
TIME+	Temps total d'utilisation du processeur depuis le lancement du processus
COMMAND	Nom de lu processus

La commande jobs

Pour connaître les processus qui s'exécutent en arrière plan.

3. Commande kill

Cette commande envoie un signal à un processus en précisant le PID du processus concerné.

kill -n°du_signal PID_du_processus

« Tuez » le processus que vous avez laissé tourner depuis le début, en utilisant cette commande kill.

Vérifiez avec top ou ps que votre processus a bien été « tué ».

kill 9 pid-processus

4. Lancer en processus en tâche de fond (arrière plan)

Pour lancer une commande quelconque, vous en saisissez le nom après le prompt du shell, tant que la commande n'est pas terminée, vous n'avez plus la main au niveau du shell, vous ne

disposez plus du prompt. Si la commande prend un certain temps, votre shell ne vous donnera pas la main tant que la commande n'est pas terminée, vous êtes obligé de lancer un autre shell, pour taper une autre commande. Vous disposez d'une technique simple qui permet de lancer une commande à partir d'un shell, et de reprendre aussitôt la main. Il vous suffit de rajouter un **&** à la fin de commande. Celle-ci se lancera en " tâche de fond ", et vous reviendrez directement au prompt du shell.

En tapant une commande en tâche de fond, vous aurez à l'affichage :

```
> ps aux &
[321]
>
```

A la suite de la saisie de la commande suivie d'un **&**, le shell vous donne immédiatement la main, et affiche le numéro du **PID** du processus lancé.

En lançant une commande à partir du shell sans le **&** à la fin, et si celle-ci prend du temps à vous rendre la main, vous pouvez faire en sorte qu'elle bascule en tâche de fond, pour que vous repreniez la main.

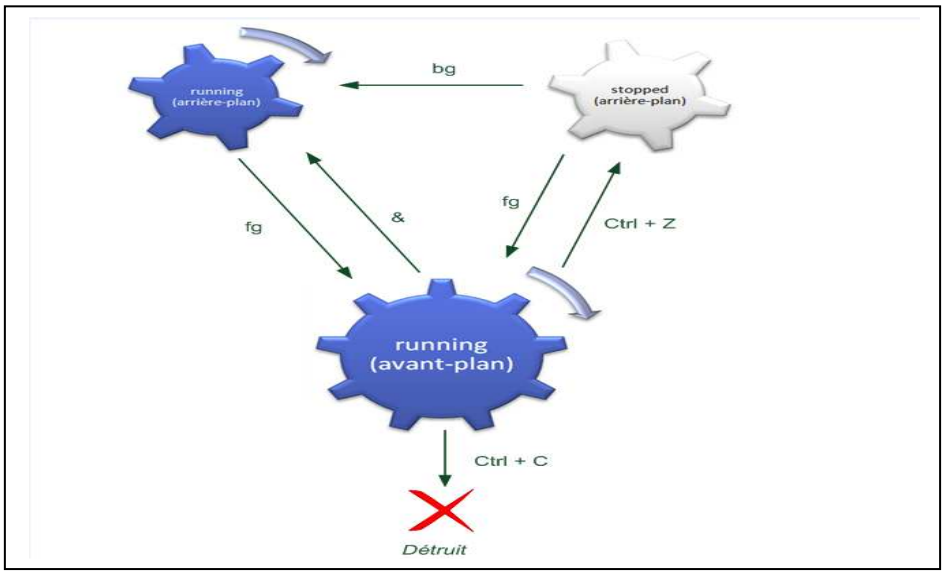
```
>top
```

Vous voulez basculer top en tâche de fond tapez, **CTRL+Z**, il va afficher

311 stopped + 311 étant le **PID** du process top. Tapez ensuite **bg** (pour background), vous voyez s'afficher : [311]

ça y est votre processus top est en tâche de fond et le shell vous rend la main.

Pour revenir en avant plan (foreground) utilisez la commande **fg**



TP N 1 : Gestion des processus

Exercice N°1

Utiliser le compilateur c sous linux GCC pour compiler les différents programmes tel que la syntaxe est comme suite :

#gcc –c nom-fichier.c

#gcc –o nom-executable nom-fichier.o

#./nom-executable

Compilez le programme suivant avec le compilateur gcc

```
#include<sys/types.h>
#include<unistd.h> //pour fork()
#include<stdio.h> //pour printf()
#include <stdlib.h>

main()
{
int i;

i=1;

while (i<=10)

{
printf("TP LINUX\n");

sleep(5);

i=i+1;

}
}
```

1- indiquez le pid de ce processus

2- Lancer en processus en tâche de fond (en arrière de plan)

3- revenir en avant plan de processus

4- mettre en pause l'exécution de ce processus (stopper le processus).

5- Arrêtez l'exécution de ce processus.

6- Utilisez la commande kill -l pour visualiser les 255 signaux prédéfinis de système linux

Partie N°2 : création des processus par la commande fork()

Exercice N°2

Q 1. Examiner les programmes et prévoir ce qui s'affichera à l'écran (sans exécuter le programme).

Q 2. Combien de processus sont-ils créés lors d'une exécution pour les trois programmes suivant ? Donnez l'arborescence de processus pour les trois programmes.

Programme 1 :

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>

void main()
{
int id, i ;

fork();
printf("A\n");
if ((id=fork()==0)) {
                     printf("B\n");
                     exit(0);
                    }
wait(0);
printf("C\n");
}
```

Programme 2 :

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>
void main()
{
int id, i ;
for (i=0 ; i<2; i++)
{
if (fork()==0) {
                  printf("A\n");
                  exit(0);
                }
}
wait(0);
printf("B\n");
}
```

Programme 3:

```
#include<sys/types.h>
#include<unistd.h> //pour fork()
#include<stdio.h> //pour printf()
#include <stdlib.h>

int main()
{
printf("Bonjour ");
if(fork () == 0)
{
printf("Monsieur\n");
exit(0);
}
else
{
printf("Madame\n");
wait(NULL);
exit(0);
}
return 0 ;
}
```

1 : Exécuter le programme 3. Expliquer pourquoi le mot `"Bonjour"` apparaît deux fois à l'écran alors qu'il n'y a qu'un seul `printf("Bonjour ");`.

2 : utilisez la fonction fflush(stdout) pour vider le buffer ou la mémoire tampon et corrigé le problème de double d'affichage de bonjour

3 : ajouter \n après bonjour et recompiler une deuxième fois le programme. Quelle est le résultat affiché par ce programme ?

Exercice N°4

1 : Ecrire un programme C qui crée deux fils, l'un affiche les entiers de 1 à 5, l'autre de 6 à 10 et le père affiche le message fin d'exécution (le père doit attendre la fin d'exécution de ces deux fils par wait()

2 : Enlevez wait() et voir la nouvelle résultat de l'affichage

Partie N°3 : Synchronisation de processus père et fils

Dans cette phase, on va utiliser les fonctions suivantes :

exit(i) :

Termine un processus, *i* est un octet (donc valeurs possibles : 0 à 255) renvoyé dans une variable du type **int** au processus père.

wait(&Etat):

Met le processus en attente de la fin de l'un de ses processus fils. Quand un processus se termine, le signal SIGCHILD est envoyé à son père. La réception de ce signal fait passer le processus père de l'état bloqué à l'état prêt. Le processus père sort donc de la fonction **wait**. La valeur de retour de **wait** est le numéro du processus fils venant de se terminer.

Lorsqu'il n'y a plus (ou pas) de processus fils dont il faut attendre la fin, la fonction **wait** renvoie -1.

Chaque fois qu'un fils se termine, le processus père sort de **wait**, et il peut consulter **Etat** pour obtenir des informations sur le fils qui vient de se terminer.

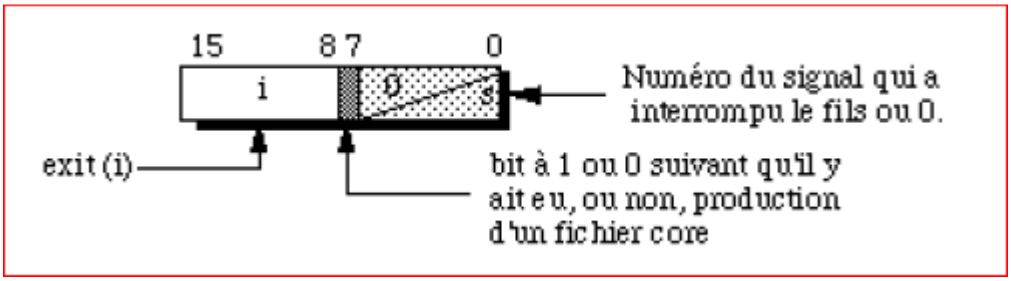
Etat est un pointeur sur un mot de deux octets :

- L'octet de poids fort contient la valeur renvoyée par le fils (i de la fonction exit(i)),
- l'octet de poids faible : contient 0 dans le cas général,

En cas de terminaison anormale du processus fils, cet octet de poids faible contient la valeur du signal reçu par le fils.

Cette valeur est augmentée de 80 en hexadécimal (128 décimal), si ce signal a entraîné la sauvegarde de l'image mémoire du processus dans un fichier core.

Schéma résumant le contenu du mot pointé par **Etat** à la sortie de **wait** :



Exercice N°5

Q1 : Compiler le programme suivant en utilisant le compilateur gcc. Indiquer le résultat afficher par ce programme.

Q2 : indiquer la relation qui existe entre exit() et wait()

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

int main(void) {
int x;
int pid;
int p ;
```

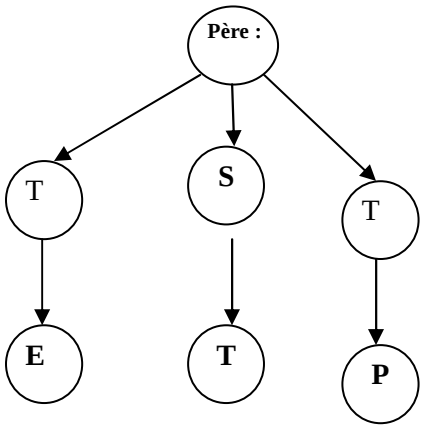
```
pid =fork();
if (pid) {
           wait(&x);
           p = x>>8;
           p = p*p;
    printf("processus pere execute  le produit de x :%d\n",p);
    exit(0);

         }
else   {
         printf("entrer la valeur de x : \n");
         scanf("%d",&x);
         exit(x); }

return 0;}
```

Exercice N°6

Ecrire un programme qui prenant une matrice de taille 2*2 et crée quatre processus fils. Chaque fils calcule un élément du carré de la matrice initiale et le renvoi au processus père comme code de retour.

Exercice N°7

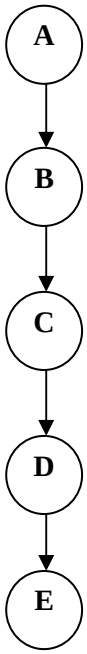
Ecrire un programme en langage c qui permettre de créer l'arborescence suivante :



Synchronisez entre les différents processus pour avoir l'affichage : **Père : TEST TP en utilisant la fonction wait(0) et exit(0)**.

Exercice N°8

Ecrire un programme en langage c qui permettre de créer l'arborescence suivante avec deux façons différents (**avec la boucle for et sans la boucle for**) **en affichant le message ABCDE**.


A
B
C
D
E