L'objectif de ce TP est d'observer les processus Linux dans tous leurs états et de créer des processus simples et parallèles.

TP N°3
(Gestion des processus Linux)

Exercice 1

1. Utiliser le manuel (commande ***man***) pour renseigner les commandes: ***trap,ps,kill,nohup,top,pstree,bg,fg***.

2. créer un programme en langage c qui fait une boucle vide infinie et lancer le processus correspondant,

3. donner son ***pid*** et l'identité et le ***pid*** de son processus père,

3. que fait chacune des combinaisons de touches suivantes: ***CTRL+C*** et ***CTRL+Z***,

4. comment remettre un processus au premier plan après un ***CTRL+Z***,

5. refaire 3 en utilisant la commande ***kill***

6. comment lancer un processus en arrière plan ? Quelle est la différence avec la combinaison ***CTRL+Z***.

Exercice 2

1. Donner des exemples de quelques signaux avec leur numéro
2. Soit le script bash suivant:

```
i= 0
while true
do
i=`expr $i + 1`
echo "Valeur: $i"
sleep 2
done
```

Éditer ce script dans un fichier nommée ex et exécuter le. Que fais ce script ?
3. Dans un shell bash, inhiber le signal 2 puis exécuter le programme ex. Essayer d'interrompre le programme par les touches CTRL+C. Quelle est la réponse du système ? Essayer les touches CTRL+Z. Quelle est la réponse du système ?
4. Activer la réaction associée au signal 2 et refaire la question 2.
5. En utilisant la commande ps, tracer le chemin dans la hiérarchie de processus qui mène du processus de shell de connexion d'un utilisateur au processus racine init. Utiliser aussi pstree.
6. Dans un shell bash lancer le programme ex puis interrompre l'exécution sen tapant CTRL+Z.
Donner deux manières pour arrêter le processus suspendu (celui correspondant au programme) en utilisant la commande kill.
7. Modifier le programme donné en exemple 3 pour que la valeur du compteur soit sauvegardée dans un fichier au lieu d'être affichée sur l'écran. Puis lancer le programme modifié de sorte qu'il survive à la déconnexion de l'utilisateur (en utilisant la commande nohup). Vérifier le fonctionnement de cette commande. Quel est le processus père du processus correspondant au programme ex?

Exercice 3

Soit le programme suivant :

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

void handler(int);
int nsig[NSIG];

int main(void)
{  int s;

   for (s = 1; s < NSIG; s++)
   {  if (signal(s, handler) == SIG_ERR)
         fprintf(stdout, "Je ne peux pas attraper signal no %d\n", s);
      nsig[s] = 0;
   }

   while (1) pause();
}


void handler(int s)
{  printf("Signal %d recu %d fois\n", s, ++nsig[s]);
}
```

1. Dérouler ce programme pour dire ce qu'il fait.

2. Comment écrire un programme pour détourner un seul signal ?

Exercice 4

Soit le code suivant écrit en c :

```
#include <stdio.h>
#include <unistd.h>
int main ()
{
printf ("L'identifiant du processus est %d\n", (int) getpid ());
printf ("L'identifiant du processus parent est %d\n", (int) getppid ());
return 0;
}
```

1. Excécuter ce programme plusieurs fois et noter les valeurs affichées ? Avez vous les mêmes valeurs pour le pid et le ppid ? Expliquer.

Exercice 5 (création de processus)

Soit le programme suivant :

```
int main ()
{
int return_value;
return_value = system ("ls -l /");
return return_value;
}
```

1. Que fait la primitive system ? Vérifier en remplaçant la commande ls par le programme de l'exercice 1.

Exercice 6 (création de processus)

Soit le programme suivant :

```
int main ()
{
pid_t child_pid;
printf ("ID de processus du programme principal : %d\n",(int)getpid ());
child_pid = fork ();
if (child_pid != 0) {
printf ("je suis le processus parent, ID : %d\n", (int) getpid ());
printf ("Identifiant du processus fils : %d\n", (int) child_pid);
}
else
printf ("je suis le processus fils, ID : %d\n", (int) getpid ());
return 0;
}
```

1. dérouler et expliquer,
2. transformer ce programme de sorte que le fils renvoie un résultat à son père.

Exercice 7 (création de processus)

Utiliser l'une des primitives de la famille exec pour lancer une commande / un programme.