Système d'exploitation
TP : Gestion des processus

1-Soit le programme suivant **:**

```
#include <unistd.h>
#define   N    .....

void main ()
{ int     i;
pid_t   pid, ppid, npid;
pid = (pid_t) getpid();
printf('' Debut du processus n° : %d  \n'', pid );
for ( i = 1 ; i < N ; i++ ) {
        npid = ( pid_t ) fork();
        pid  = (pid_t) getpid();
        ppid  = (pid_t) getppid();
        printf(''Le processus est %d , le père  est : %d'', pid,ppid) ;
        }
printf('' \t  \t  \t  Fin du processus n° : %d  \n'', pid );
}
```

Combien de processus sont créés pour N=2, N=3, N=4, ... ? Donner l'arborescence des processus.

2.1- Ecrire en C un programme appelé « creer_fils » qui crée un processus fils. Si la création du fils échoue le programme doit l'indiquer par un message d'erreur. Les processus père et fils doivent afficher leurs numéros d'identité ainsi que ceux de leurs pères.

2.2- Modifier le programme « creer_fils » en utilisant **sleep()** ou **wait()**, pour que
- l'activité propre au processus fils aie lieu **avant** celle du processus père,
- l'activité propre au processus fils aie lieu **après** celle du processus père.

Commenter les différences entre les solutions utilisant sleep() et wait(), après plusieurs exécutions du programme en premier plan et en arrière plan. Constater le rattachement des processus zombies au processus init (de pid 1) en utilisant la commande ps dans le cas d'exécution en arrière plan.

2.3- Modifier le programme « creer_fils » en declarant dans le processus père une variable initialisée à une valeur choisie. Cette variable sera affichée avant et après la terminaison du fils qui modifie cette variable. Qu'est ce que vous déduisez d'après l'observation à propos du mécanisme d'héritage des variables.

2.4- Ecrire un programme « **affiche100** »qui affiche 100 fois la chaîne de caractère reçue en argument (à la ligne de commande). Exemple :

```
$ affiche100  a
aaaaaaaaaaaaaa.....aaaaa   ( 100 fois)
```

- Modifier le programme « creer_fils » pour que le programme affiche100 soit lancé dans le processus fils. Utilisez l'appel execlp().
- Modifier le programme« creer_fils » pour que le père lance un deuxième fils qui affiche son numéro d'identité. Le deuxième fils doit être lancé sans attendre la fin du premier fils pour que les 3 processus soient concurrents.

3- Ecrire un programme appelé « loop » qui crée un processus fils. Les deux processus père et fils exécutent une boucle sans fin. Le caractère interruption entré au clavier du terminal provoque l'envoie d'un signal aux 2 processus. On veut observer la réaction des deux processus à l'envoie de ce signal dans les deux cas ou loop est lancé en premier plan et en arrière plan. Utilisez la commande « ps -l » pour observer le comportement des processus lancés en arrière plan. Utiliser la commande « kill -num pid » pour envoyer le signal de numero *num* au processus d'identité *pid*. Qu'est ce qui se passe si vous envoyez au père et au fils les signaux SIGINT (n° 2), SIGKILL (n° 9) et SIGFPE (floating exception de n° 8) ?

4.1- Ecrire un programme qui crée un processus fils qui affiche un message quand il reçoit le signal SIGUSR1 du processus père.
4.2- Ecrire un programme qui crée un fils. Le père attendra la terminaison de son fils et affichera la raison de mort du processus fils. Tester votre programme en tuant le fils depuis le terminal.