TP n°3

Exercice 1 (sur papier)

Soit le programme suivant :

```
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<sys/wait.h>
4
5  int main()
6  {
7         int i, retourExit, retourFork, pid;
8
9         printf("Le shell a pour PID %d\n", getppid());
10        for(i=0; i < 3; i++)
11        {
12                retourFork=fork();
13                if (retourFork > 0) {
14                        printf("Je viens de me forker ! Mon PID est %d mon Pere est %d et
15 i=%d\n",getpid(),getppid(),i);
16                } else {
17                        printf("Je suis nouveau ! Mon PID est %d mon Pere est %d et i=%d\
18 n",getpid(),getppid(),i);
19                }
20        }
21
22        while ((pid = wait(&retourExit)) > 0) {
23                printf("Mon fils %d vient de se terminer\n", pid);
24        }
25
26        return i;
27 }
```

Voici ce que produit une exécution de ce code :

```
Le shell a pour PID 13368
Je viens de me forker ! Mon PID est 15041 mon Pere est 13368 et i=0
Je suis nouveau ! Mon PID est 15042 mon Pere est 15041 et i=0
Je viens de me forker ! Mon PID est 15041 mon Pere est 13368 et i=1
Je suis nouveau ! Mon PID est 15043 mon Pere est 15041 et i=1
Je viens de me forker ! Mon PID est 15042 mon Pere est 15041 et i=1
Je viens de me forker ! Mon PID est 15041 mon Pere est 13368 et i=2
Je viens de me forker ! Mon PID est 15043 mon Pere est 15041 et i=2
Je viens de me forker ! Mon PID est 15042 mon Pere est 15041 et i=2
Je suis nouveau ! Mon PID est 15045 mon Pere est 15041 et i=2
Je suis nouveau ! Mon PID est 15046 mon Pere est 15043 et i=2
Je suis nouveau ! Mon PID est 15044 mon Pere est 15042 et i=1
Je suis nouveau ! Mon PID est 15047 mon Pere est 15042 et i=2
Mon fils 15046 vient de se terminer
Mon fils 15045 vient de se terminer
Mon fils 15047 vient de se terminer
Je viens de me forker ! Mon PID est 15044 mon Pere est 15042 et i=2
Je suis nouveau ! Mon PID est 15048 mon Pere est 15044 et i=2
Mon fils 15043 vient de se terminer
Mon fils 15048 vient de se terminer
Mon fils 15044 vient de se terminer
Mon fils 15042 vient de se terminer
```

1) Dessinez l'arbre des processus créés chronologiquement avec valeur de pid et i à chaque printf

2) Combien de processus ont-ils été créés ?

3) Recopiez, compilez et exécutez le code précédent afin de vérifier. Vous pouvez également rajouter un appel à **`sleep(30)`** à la ligne 21, qui va mettre en pause le processus pendant 30 secondes, et, pendant l'exécution de cette pause, exécuter le programme **`pstree`** (à partir d'un autre terminal) qui affiche la hiérarchie des processus en cours d'exécution sur la machine.

Exercice 2

On vous donne la fonction suivante, qui permet d'écrire une chaîne de caractères dans un fichier, caractère par caractère :

```
//writes length characters from str to the file corresponding to fd
void affiche(int fd, char* str, int length) {
      for (int i = 0 ; i < length ; i++)
            write(fd, str+i, 1);
}
```

Écrire un programme qui ouvre un fichier texte (qui n'existe pas forcément : le créer) en écriture, puis se fork, et, en utilisant la fonction précédente :

- le père écrit une suite d'une cinquantaine de lettres sur le fichier ouvert
- le fils écrit une suite d'une cinquantaine de chiffres sur le fichier ouvert

Observez le résultat produit, et expliquez ce qui s'est passé.

Dans un second temps, déplacez l'ouverture du fichier après le fork (mais de telle sorte que le père et le fils ouvrent le fichier). Observez le résultat, et expliquez ce qui s'est passé.

Exercice 3

Écrivez un programme **`zombie`** qui crée un processus qui reste zombie pendant 30 secondes.

Pour cela, il faut d'abord effectuer un fork. Puis, le père et le fils devront exécuter deux codes différents :

- le père se met en sommeil pendant 30 secondes (grâce à la fonction sleep)
- le fils se termine (avec un return)

Pour obtenir la liste des processus en mode zombie ("**`defunct`**" sous Linux) en cours d'"exécution", vous pouvez par exemple exécuter la commande suivante (depuis un autre terminal) :

```
ps –e | grep "defunct"
```

Exercice 4

Écrivez un programme **`orphelin`** qui crée un processus qui devient orphelin. Votre programme devra montrer que le processus fils perd effectivement à un moment donné son processus père, et afficher quel est son nouveau père.

Pour cela, il faut d'abord effectuer un fork. Puis, le père et le fils devront exécuter deux codes différents :

- le père affiche son process id (avec la fonction **`getpid()`**), se met en sommeil 5 secondes, puis se termine
- le fils affiche son parent id (avec la fonction **`getppid()`**), se met en sommeil 10 secondes, puis affiche à nouveau son parent id