
Ecrivez un code où un processus créé un fils. Chacun affiche son pid et le pid de son père. Le père attend son fils.

Ecrivez un programme qui créé les hiérarchies de processus suivants :

P1 → F1→F2

P1 →F1 et P1→F2

Ecrivez un programme `prog1.c` qui répond aux points suivants :

- un processus père
  - déclare une variable j et il l'initialise à 15.
  - Il crée ensuite un fils.
  - Il incrémente j de 10 unités
  - Il créé un second fils
- Le père récupère la terminaison de ses fils, puis il incrémente j de 6 unités et affiche la valeur de j.

- Chaque fils affiche son pid puis celui de son père, puis il incrémente la variable j de 5 unités pour le fils 1 et de 10 unités pour le fils 2. Chaque fils affiche la valeur de j, les fils se terminent ensuite.

Quelles sont les valeurs de j affichées par les différents processus ? Dans quel ordre à votre avis ?

Un processus parent créé 3 processus fils et attende la terminaison de chacun d'eux dans l'ordre inverse de leur création à l'aide de la primitive waitpid() ;

La fonction `wait()` ne permet pas à un processus parent d'attendre la terminaison d'un fils en particulier, en effet le processus parent reprend l'exécution de son programme lors de la réception du signal `SIGCHLD` issu de la terminaison de n'importe lequel(s) de ses fils.

Dans le cas où l'on désire attendre la terminaison d'un fils en particulier, il faut utiliser la fonction `waitpid()` cei-dessous :

```
pid_t waitpid(pid_t pid, int *status, int options);
```

Cette fonction prend en premier paramètre l'identifiant du processus fils dont la terminaison doit être prise en compte et place dans le pointeur `status` en deuxièmte paramètre le code de retour renvoyé par

le fils et retourne l'identifiant du processus fils correspondant. Le dernier paramètre permet d'utiliser des options particulières.

Voici un extrait de code :

```
int main(void)
{
  int i;
  i = 10;
  fork();
  i = i + 1;
  fork();
  i = i + 2;
}
```

Expliquez combien de processus sont créés et quelle est la hiérarchie de ces processus.

Il est possible de demander au système de remplacer tout ou partie du code à exécuter par un processus en utilisant une primitive de recouvrement `exec`. Il en existe plusieurs et elles se distinguent par la façon de récupérer les paramètres à utiliser :

- Sous forme de liste : execl, execlp, execle,
- Sous forme de tableau : execv, execvp, execve.

Par exemple, les arguments peuvent être passés sous forme de liste (**l** pour *list* en anglais), ou la variable d'environnement `PATH` est utilisée pour le chemin d'accès vers le programme exécutable à exécuter (**p** pour *path* en anglais), ou encore par modification de l'environnement (**e** pour *environment*).

Soit le programme suivant ; que réalise t-il ?

```
 #include <stdio.h>

#include <unistd.h>

 int main(void)

 {

     int pid;

     pid = fork();

     if(pid == 0)
```

```
    {

    printf("je suis le fils et vais exécuter la commande ls\n");
    execlp("ls", "ls", "-l", NULL);

    }

    else

    {

    printf("je suis le père\n");

    wait();

    }
    exit(0);

}
```

Modifiez le pour qu'il exécute la commande ps –aux ;