

Système d'exploitation II

TP 3 : Création Multiple et gestion des processus

Partie II : Création Multiples et gestion des processus

Dans cette partie du TP nous traiterons la création multiple des processus, c'est-à-dire la mise en place d'une hiérarchie des processus sous linux. Avant d'étudier les procédures de gestion des permissions et d'envoi des signaux

Question 1 : Créer un programme C qui permet :

- La création de l'arborescence pour chaque figure
- L'affichage des attributs des processus

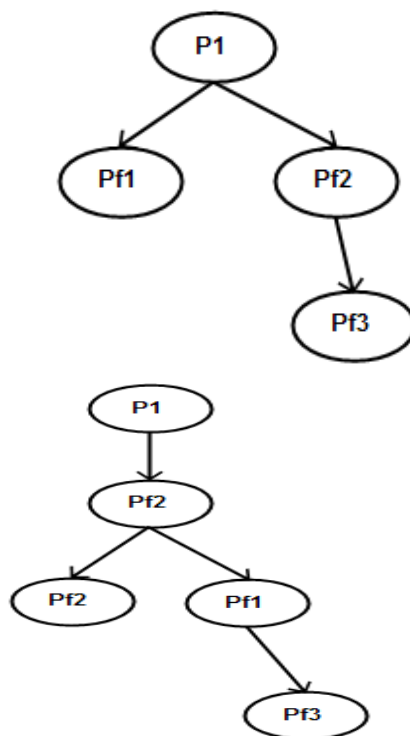


Figure 1 : Arborescence N°1

Figure2 : Arborescence N°2

Solution

Dans les cas d'une grande hiérarchie des processus on peut simplifier le code en traitant juste le `if(fork()==0)`. C'est vrai que lors de l'exécution on peut avoir une répétition des processus mais c'est acceptable.

Figure 1

```
#include<sys/signal.h>
void attributs()
{ printf (« PID = %d PPID =%d UID=%d GID=%d »,getpid(),getppid(),getuid(),getgid() );}
```

```

int main()
{ // le processus P1 est le main
  int pf1, pf2, pf3 ;
  attributs() ;
  if(pf1=fork()==0)
    {attributs() ;}
  if(pf2=fork()==0)
    { attributs();
    if (pf3=fork()==0) {attributs();}}}

```

Figure 2

On met le fils de pf1 est pf12

```

#include<sys/signal.h>
void attributs()
{ printf (« PID = %d PPID =%d UID=%d GID=%d »,getpid(),getppid(),getuid(),getgid() );}
int main()
{ // le processus P1 est le main
  int pf1, pf2, pf3 ; pf12
  attributs() ;
  if(pf2=fork()==0)
    {attributs() ;
    if(pf12=fork()==0) {attributs();}
    if(pf1=fork()==0)
      { attributs();
      if(pf3=fork()==0) {attributs();}
      }
    }
}

```

Question 2 : Les priorités.

- Est-ce qu'on peut connaître à l'avance l'ordre d'exécution ? Expliquer.

Non on ne peut pas savoir l'ordre car les processus ont la même priorité (0). Donc l'ordre dépendra de la technique utilisée.

Sous linux on peut définir les priorités via la fonction **nice** et **renice**.

- **nice niveau commande** Pour définir la priorité d'un processus (une commande) avant son lancement.
- **renice niveau PID** Pour redéfinir la priorité d'un processus en cours d'exécution.

Avec, niveau est un entier qui définit la priorité, il prend des valeurs entre [-20, 20], un nombre plus petit signifie une priorité plus grande. Les valeurs négatives sont réservées à l'utilisateur root.

- Modifier le code pour que les processus attendent un instant avant de terminer l'exécution (utiliser la fonction `sleep()`).

Il suffit d'ajouter la fonction `sleep()` après la fonction attributs dans la partie de chaque processus

- Rendre les priorités de $PF3 > PF2 > PF1 > P$ de la figure 1. Exécuter et expliquer.

Pour changer la priorité, il faut lancer le programme dans un terminal, ouvrir un deuxième terminal et utiliser la commande `renice` avec le PID affiché. L'indice de priorité de PF3 doit être plus petit que celui de PF2 et ainsi de suite.

Question 3 : Arrêt d'un processus en exécution

En pratique, on est mené à tuer des processus en exécution pour une raison ou une autre. Pour cela, il suffit d'envoyer un signal au processus via la commande **kill**, qui permet l'envoi d'un signal au processus visé (utilisation du PID). La syntaxe est : **# kill -signal process-id**

Plusieurs types de signal existent sous linux parmi lesquelles :

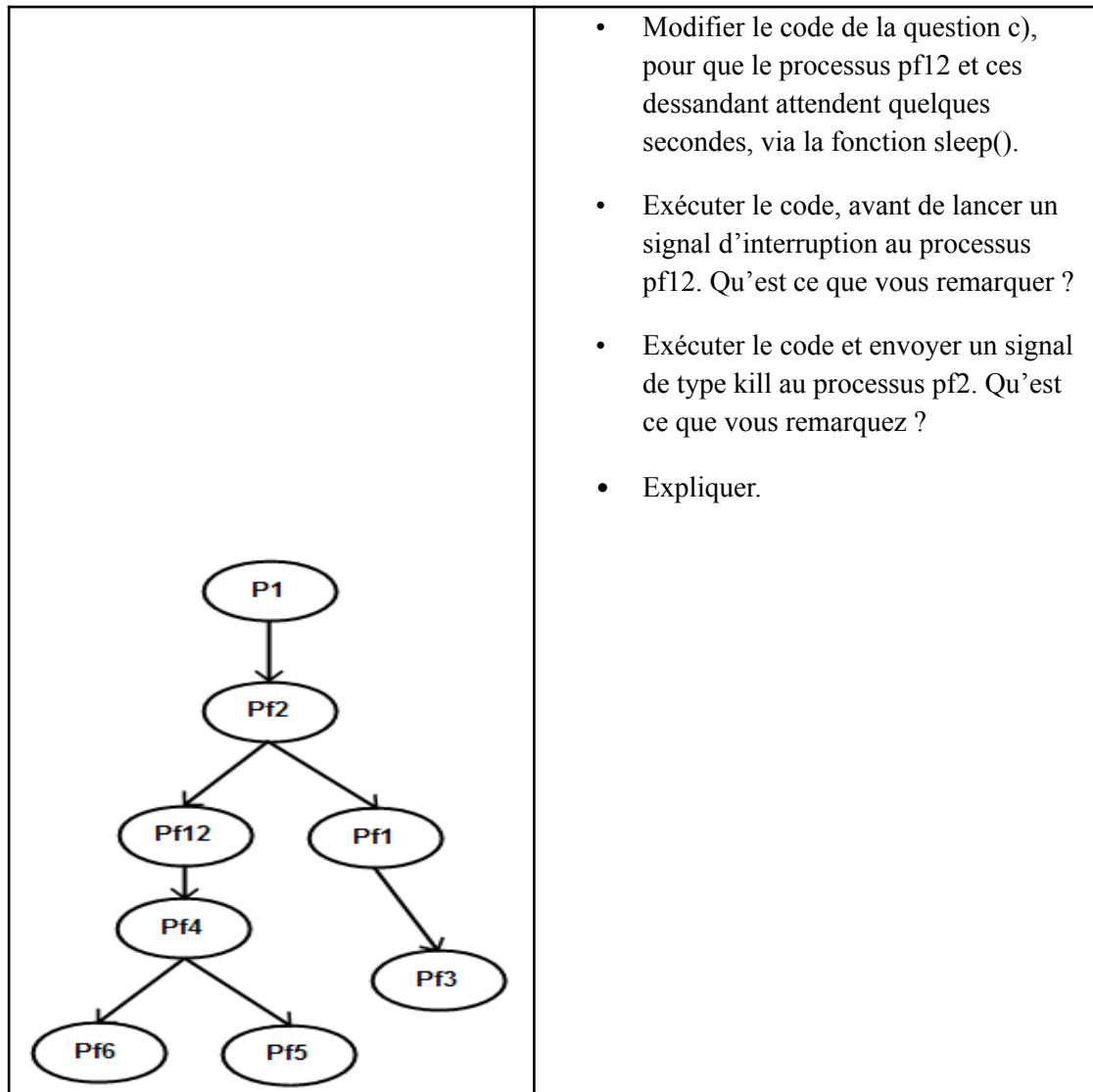
- a) Définissez un signal. C'est quoi la différence entre les signaux et les pipes ? Donner un exemple de signal.

Un signal c'est un message prédéfini qui permet de provoquer une action. Les pipes permettent l'envoi des messages, tandis que les signaux permettent de provoquer une action. Exemple de signal `SIGKILL`, `SIGTERM`

- b) Lister les signaux disponibles via la commande `kill`. Utiliser l'option `-l`.

Kill -l : 64 signaux seront affichés (voire le cours vidéo)

- c) Créer un programme qui permet la création de l'arborescence 3, en affichant pour chaque processus son PID.



c) Le programme :

```

// P1 est le main
#include<sys/signal.h>
void attributs()
{ printf (« PID = %d PPID =%d »,getpid(),getppid());}
int main()
{ // le processus P1 est le main
  int pf2, pf12, pf4 ; pf6,pf5,pf1,pf3;
  attributs() ;
  if(pf2=fork()==0)
  {attributs() ;
    if(pf1=fork()==0)
    {attributs();
      if(pf3=fork()==0) { attributs();}
    }
    if(pf12=fork()==0)

```

```

        {attributs();
        if(pf4=fork()==0)
            {attributs();
            if(pf5=fork()==0) { attributs();}
            if(pf6=fork()==0) { attributs();}
            }
        }
    }
}

```

- Modifier le code de la question c), pour que le processus pf12 et ses descendants attendent quelques secondes, via la fonction sleep().

Il suffit d'ajouter la fonction sleep(20) après la fonction attributs().

- Exécuter le code, avant de lancer un signal d'interruption au processus pf12. Qu'est-ce que vous remarquez ?

Le système tuera tous les descendants de pf12 notamment pf4, pf5 et pf6

- Exécuter le code et envoyer un signal de type kill au processus pf2. Qu'est-ce que vous remarquez ?

Le système tuera tous les descendants de pf2, donc tous les processus seront tués sauf P.

- Expliquer.

Le système tue tous les descendants du processus pour éviter l'état zombis.