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

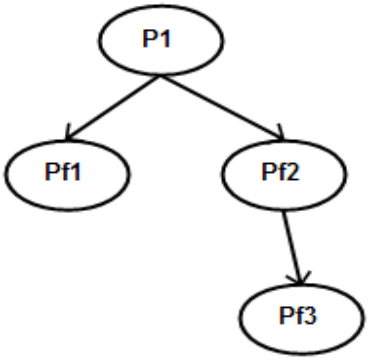


Figure 1 : Arborescence N°1

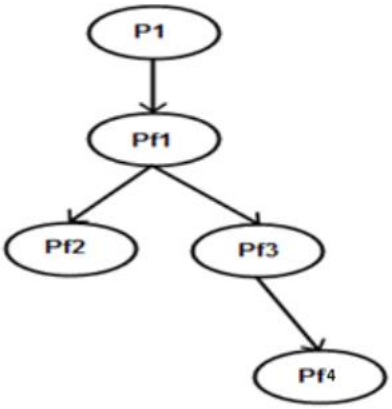


Figure2 : Arborescence N°2

Question 2 : Les priorités.

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

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.

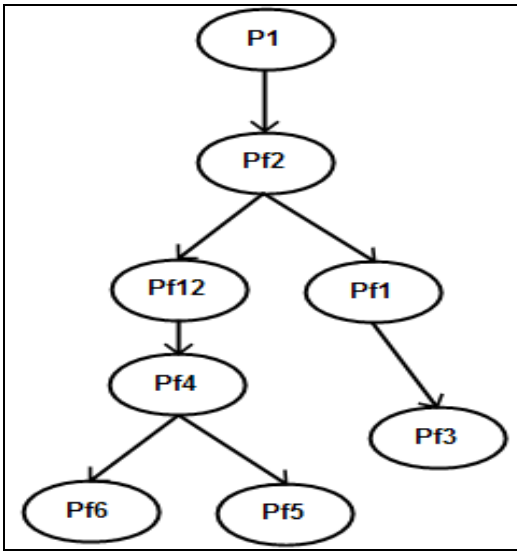
- Modifier le code pour que les processus attendent un instant avant de terminer l'exécution (utiliser la fonction sleep()).
- Rendre les priorités de PF3>PF2>PF1>P de la figure 1. Exécuter et expliquer.

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.
b) Lister les signaux disponibles via la commande kill. Utiliser l'option –l.
c) Créer un programme qui permet la création de l'arborescence 3, en affichant pour chaque processus son PID.



- Modifier le code de la question c), pour que le processus pf12 et ces descendant attendent quelques secondes, via la fonction sleep().
- Exécuter le code, avant de lancer un signal d'interruption au processus pf12. Qu'est ce que vous remarquer ?
- Exécuter le code et envoyer un signal de type kill au processus pf2. Qu'est ce que vous remarquez ?
- Expliquer.

==========Solution ==========

Dans le 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.

RQ : On considère ici que P1 est le processus main

Figure 1

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

void attributs(char *nom)
{
    printf ("Nom %s : PID = %d PPID =%d UID=%d GID=%d\n",nom, getpid(),
getppid(), getuid(), getgid());
}

int main()
{
    // le processus P1 est le main
```

```
    int pf1, pf2, pf3;
    attributs("P1") ;
    if((pf1=fork())==0)
    {
        attributs("Pf1");
        exit(0);
    }

    if((pf2=fork())==0)
    {
        attributs("Pf2");

        if ((pf3=fork())==0)
        {
            attributs("Pf3");
            exit(0);
        }
        exit(0);
    }
    return 0;
}
```

Figure 2

On met le fils de pf1 est pf12

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

void attributs(char* nom)
{
    printf ("Nom %s : PID = %d PPID =%d UID=%d GID=%d\n", nom, getpid(),
getppid(), getuid(), getgid());
}

int main()
{
    // le processus P1 est le main
    int pf1, pf2, pf3, pf4;
    attributs("P1") ;
    if((pf1=fork())==0)
    {
        attributs("Pf1") ;
        if((pf2=fork())==0)
        {
            attributs("Pf2");
            exit(0);
```

```
        }
        if((pf3=fork())==0)
        {
            attributs("Pf3");
            if((pf4=fork())==0)
            {
                attributs("Pf4");
                exit(0);
            }
            exit(0);
        }
    }
    return 0;
}
```

Question 2 : Les priorités.

- Est-ce qu'on peut connaitre 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 d'ordonnancement utilisée.

- 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(10) 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

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 prédéfinit 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, SIGTERMINATE ….

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

kill -l : 64 signaux seront affichés

```
ouassit@Youssef:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

c) Le programme :

```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>

void attributs(char *nom)
{
    printf ("Nom %s : PID = %d PPID =%d\n", nom, getpid(), getppid());
}

int main()
{   // le processus P1 est le main
    int pf1, pf3, pf2, pf12, pf4, pf6, pf5;
    attributs("P1");
    if((pf2=fork())==0) {
        attributs("Pf2");
        if((pf1=fork())==0) {
            attributs("Pf1");
            if((pf3=fork())==0) {
                attributs("Pf3");
            }
        } else if((pf12=fork())==0) {
            attributs("Pf12");
            if((pf4=fork())==0) {
                attributs("Pf4");
                if((pf5=fork())==0) {
                    attributs("Pf5");
                } else if((pf6=fork())==0) {
                    attributs("Pf6");
                }
            }
        }
    }
    return 0 ;
}
```

```
   Ou :

int main()
{   // le processus P1 est le main
    int pf1, pf3, pf2, pf12, pf4, pf6, pf5;
    attributs("P1");
    if((pf2=fork())==0) {
        attributs("Pf2");
        if((pf1=fork())==0) {
            attributs("Pf1");
            if((pf3=fork())==0) {
                attributs("Pf3");
                exit(0);
            }
            exit(0);
        }
        if((pf12=fork())==0) {
            attributs("Pf12");
            if((pf4=fork())==0) {
                attributs("Pf4");
                if((pf5=fork())==0) {
                    attributs("Pf5");
                    exit(0);
                }
                if((pf6=fork())==0) {
                    attributs("Pf6");
                    exit(0);
                }
            }
        }
        exit(0);
    }
    return 0;
}
```

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

Il suffit d'ajouter la fonction sleep(20) après l'appel de fonction attributs("Pf5") et attributs("Pf6").

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

Par défaut, lorsqu'un processus parent est tué, ses processus enfants ne sont pas automatiquement supprimés. Au lieu de cela, ils deviennent « orphelins ».

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

Tous les descendants de pf2 deviennent « orphelins » et seront adoptés par le processus init.

Expliquer.

Ces processus orphelins sont ensuite adoptés par le processus init (ou un autre processus système comme systemd dans les systèmes Linux modernes), qui devient alors leur nouveau parent. Le processus système d'initialisation est responsable du nettoyage des processus et de la récolte des processus zombies (processus dont l'exécution est terminée mais qui détiennent toujours une entrée dans la table des processus).