EXERCICES DIRIGES – TRAVAUX PRATIQUES
Mécanismes de base sous Linux
Processus Linux (Fork, wait, exit)
Primitives de recouvrement

PROCESSUS LINUX

PREAMBULE

La création d'un tel type de processus est réalisée à l'aide la primitive `fork()` suivante :

```
pid_t fork(void);
```

Les fonctions getpid() et getppid() permettent à un processus d'obtenir respectivement son identifiant et celui de son processus parent :

```
pid_t getpid(void);

pid_t getppid(void);
```

Pour permettre à un processus parent de prendre en compte la terminaison (réception du signal `SIGCHLD`) d'un (quelconque) processus, il est nécessaire d'utiliser la fonction `wait()` ci-dessous :

```
pid_t wait(int *status);
```

Cette fonction est bloquante, c'est-à-dire qu'à l'appel de cette fonction le processus est mis en sommeil (progression de l'exécution gelée) jusqu'à la réception du signal `SIGCHLD`.
Cette fonction place dans le pointeur passé en paramètre le code de retour renvoyé par le fils et retourne l'identifiant du processus fils correspondant.

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.

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*).

La famille Exec est constituée de 6 primitives dont les prototypes sont donnés ci-après :
– int execl (const char *ref, const char *arg, …, NULL): ref est le chemin d'un exécutable à partir du répertoire courant, const char *arg, …, NULL est la liste des arguments.
– int execlp (const char *ref, const char arg, …, NULL): ref est le chemin d'un exécutable à partir de la variable d'environnement PATH, const char *arg, …, NULL est la liste des arguments.
– int execle (const char *ref, const char *arg, …, const char *envp[]): ref est le chemin d'un exécutable à partir du répertoire courant, const char *arg, …, NULL est la liste des arguments, const char *envp[] est un tableau spécifiant les variables d'environnement sous la forme nom_var = valeur.

EXERCICE 1

Soit le programme `essai.c` suivant:

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

main()
{
int pid;
pid = fork();
if (pid == 0)
     {for(;;)
      {printf ("je suis le fils\n"); exit(0);}
      }
else
     {for(;;)
      printf("je suis le père\n");
     }
}
```

Question 1 – On compile ce programme pour générer un exécutable appelé `essai` dont on lance l'exécution. La commande `ps -l` permettant d'afficher les caractéristiques de l'ensemble des processus de l'utilisateur donne les éléments suivants:

```
F   S UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY    TIME    CMD
100 S  0   467   1    0  60   0 - 256 read_c tty5   00:00:00 mingetty
100 S  0   468   1    0  60   0 - 256 read_c tty6   00:00:00 mingetty
100 S  0   576   570  0  60   0 - 498 read_c pts/0  00:00:00 cat
100 S  0   580   578  0  70   0 - 576 wait4  pts/1  00:00:00 bash
100 S  0   581   579  0  60   0 - 77 wait4  pts/2   00:00:00 bash
000 S  0   592   581  2  61   0 - 253 down_f pts/2  00:00:01 essai
```

```
040 S  0   593   592  2  61   0 - 253 write_ pts/2  00:00:01 essai
100 R  0   599   580  0  73   0 - 652 -      pts/1  00:00:00 ps
```

Les champs S, PID, PPID et CMD codent respectivement l'état du processus (S pour *Stopped*, R pour *Running*), la valeur du PID et du PPID pour le processus et le nom du programme exécuté. On tape la commande `kill -9 593` qui entraîne la terminaison du processus dont le pid 593 est spécifié en argument. L'exécution de la commande `ps -l` donne à présent le résultat suivant. Que pouvez-vous dire à ce sujet?

```
F   S UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY    TIME    CMD
100 S  0   467   1    0  60   0 - 256 read_c tty5   00:00:00 mingetty
100 S  0   468   1    0  60   0 - 256 read_c tty6   00:00:00 mingetty
100 S  0   576   570  0  60   0 - 498 read_c pts/0  00:00:00 cat
100 S  0   580   578  0  70   0 - 576 wait4  pts/1  00:00:00 bash
100 S  0   581   579  0  60   0 - 77 wait4  pts/2   00:00:00 bash
000 S  0   592   581  2  60   0 - 253 write_ pts/2  00:00:03 essai
444 Z  0   593   592  2  60   0 - 0 do_exi pts/2    00:00:03 essai
<zombie>
100 R  0   601   580  0  73   0 - 651 -      pts/1  00:00:00 ps
```

Question 2 – On lance de nouveau l'exécution du programme `essai`. La commande `ps -l` permettant d'afficher les caractéristiques de l'ensemble des processus de l'utilisateur donne les éléments suivants:

```
F   S UID PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY  TIME    CMD
100 S  0  467  1    0  60   0 -   256 read_c tty5   00:00:00 mingetty
100 S  0  468  1    0  60   0 -   256 read_c tty6   00:00:00 mingetty
100 S  0  576  570  0  60   0 -   498 read_c pts/0  00:00:00 cat
100 S  0  580  578  0  65   0 -   576 wait4  pts/1  00:00:00 bash
100 S  0  581  579  0  60   0 -   577 wait4  pts/2  00:00:00 bash
000 S  0  604  581  3  60   0 -   253 write_ pts/2  00:00:00 essai
040 S  0  605  604  2  60   0 -   253 down_f pts/2  00:00:00 essai
100 R  0  606  580  0  76   0 -   649 -      pts/1  00:00:00 ps
```

On tape la commande `kill -9 604` qui entraîne la terminaison du processus dont le pid 604 est spécifié en argument. L'exécution de la commande `ps -l` donne à présent le résultat suivant. Que pouvez-vous dire à ce sujet?

```
F   S UID PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY    TIME   CMD
100 S  0  467  1    0  60   0 -   256 read_c tty5   00:00:00 mingetty
100 S  0  468  1    0  60   0 -   256 read_c tty6   00:00:00 mingetty
100 S  0  576  570  0  60   0 -   498 read_c pts/0  00:00:00 cat
100 S  0  580  578  0  65   0 -   576 wait4  pts/1  00:00:00 bash
100 S  0  581  579  0  60   0 -   577 wait4  pts/2  00:00:00 bash
100 S  0  581  579  0  60   0 -   577 read_c pts/2  00:00:00 bash
040 S  0  605  1    2  60   0 -   253 write_ pts/2  00:00:02 essai
100 R  0  608  580  0  73   0 -   649 -      pts/1  00:00:00 ps
```

Question 3 – Le programme `essai.c` est modifié comme ci-dessous:

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

```
main()
{
int pid;
pid = fork();
if (pid == 0)
     {for(;;)
      {printf ("je suis le fils\n"); exit(0);}
      }
else
     {
      printf("je suis le père\n");
      wait();
     }
}
```

On recompile ce programme pour générer un nouvel exécutable appelé `essai` dont on lance l'exécution. La commande `ps -l` permettant d'afficher les caractéristiques de l'ensemble des processus de l'utilisateur donne les éléments suivants:

```
F   S UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY    TIME    CMD
100 S  0  467  1     0  60   0 -   256 read_c tty5 00:00:00  mingetty
100 S  0  468  1     0  60   0 -   256 read_c tty6 00:00:00  mingetty
100 S  0  576  570   0  60   0 -   498 read_c pts/0  00:00:00 cat
100 S  0  580  578   0  70   0 -   577 wait4  pts/1  00:00:00 bash
100 S  0  581  579   0  60   0 -   577 wait4  pts/2  00:00:00 bash
000 S  0  627  581   0  60   0 -   253 wait4  pts/2  00:00:00 essai
040 S  0  628  627   3  61   0 -   253 write_ pts/2  00:00:00 essai
100 R  0  629  580   0  72   0 -   649 -      pts/1  00:00:00 ps
```

On tape la commande `kill -9 628` qui entraîne la terminaison du processus dont le pid 628 est spécifié en argument. L'exécution de la commande `ps -l` donne à présent le résultat suivant. Que pouvez-vous dire à ce sujet?

```
F   S UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY    TIME     CMD
100 S  0  467  1     0  60   0 -   256 read_c tty5 00:00:00  mingetty
100 S  0  468  1     0  60   0 -   256 read_c tty6 00:00:00  mingetty
100 S  0  576  570   0  60   0 -   498 read_c pts/0   00:00:00 cat
100 S  0  580  578   0  70   0 -   577 wait4  pts/1   00:00:00 bash
100 S  0  581  579   0  60   0 -   577 wait4  pts/2   00:00:00 bash
100 R  0  31   580   0  73   0 -   648 -      pts/1   00:00:00 ps
```

EXERCICE 2

En reprenant le code de l'exercice 1, ecrivez un programme où un processus crée un fils, le fils affiche son pid et celui de son père, le père affiche son pid et celui de son fils. Il attend la fin de son fils. Utilisez le squelette disponible sur deptinfo.cnam.fr.

Affichage d'un entier i : `printf ("valeur de i %d\n", i);`

EXERCICE 3

Modifiez le programme précédent pour que le fils recouvre son code et exécute la commande `ls -l`.

EXERCICE 4

Ecrivez un programme de sorte à ce que le processus parent créé 3 processus fils et attende la terminaison de chacun d'eux dans l'ordre inverse de leur création. Chaque fils créé affiche son pid. Pour cela vous aurez besoin de créer un tableau de 3 entiers afin de stocker l'identifiant de chaque fils créé avec la fonction fork().

Exercice 2

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

int main(void)
{
     int pid, i;

     pid = fork();
     if(pid == 0)
     {
        printf("je suis le fils d'identifiant %d et le processus %d est
     mon père\n", getpid(), getppid());
        exit(0);
     }
     else
     {
         {printf("je suis le père d'identifiant %d et le processus %d est
     mon fils\n", getpid(), pid);}
          wait();

     }
}
```

Exercice 3

```
#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);
}
```