**EXERCICES DIRIGES – TRAVAUX PRATIQUES**
**Mécanismes de base sous Linux**
**Processus Linux (Fork, wait, exit)**
**Primitives de recouvrement**
**Tubes anonymes**
**MSQ**
**Socket**

# PROCESSUS LINUX

## PREAMBULE

Rappelez le mécanisme de création d'un processus Linux par le biais de la primitive fork ; le rôle de la primitive exit, wait et exec.

## EXERCICE 1

Soit le programme `essai.c` suivant:

```
#include <stdio.h>

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

main()
```

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

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`.

# TUBES ANONYMES

**PREAMBULE**
Rappelez le mécanisme de fonctionnement des tubes et les primitives associées.

**EXERCICE 4**
On souhaite réaliser une communication inter processus dans laquelle deux processus A et B s'échangent une chaîne de caractères, plus précisément :

- le processus A envoie la chaîne "hello, je suis le processus A";
- le processus B répond par la chaîne "hello, je suis le processus B".

La communication s'effectue par tubes anonymes. Ecrivez les programmes correspondants à partir des squelettes disponibles sur deptinfo.cnam.fr
.
.
**EXERCICE 5**
Soit le programme C suivant

```
#include <stdio.h>

int pip[2];

main()
{
int nb_ecrit;
int pid;


/*  ouverture d'un pipe */
if(pipe(pip))
  { perror("pipe");
  exit(1);}

pid = fork();
if (pid == 0)
{
close(pip[0]);
close(pip[1]);
printf("Je suis le fils\n");
exit();
}
else
{
close(pip[0]);
for(;;){
if ((nb_ecrit = write(pip[1], "ABC", 3)) == -1)
      {
      perror ("pb write");
      exit();
      }
else
```

```
        printf ("retour du write : %d\n", nb_ecrit);
}
}
}
```

**Question 1**
Que va-t-il se passer ?

**Question 2**
Ecrivez un handler permettant de prendre en compte le signal envoyé au processus par le système.

# MSQ

**PREAMBULE**
Rappelez le mécanisme de fonctionnement des MSQ et les primitives associées.

**EXERCICE 6**

On considère une application constituée d'un serveur et de n clients communiquant au travers d'une file de messages. Le serveur effectue l'addition de deux nombres entiers envoyés par un client et lui renvoie le résultat.

Ecrivez les programmes serveur et client correspondants à partir des squelettes disponibles sur deptinfo.cnam.fr

# SOCKET

**EXERCICE 7 : Première communication socket**
On désire réaliser un serveur itératif gérant des connexions de clients en mode connecté à l'aide de Socket TCP. A chaque nouvelle connexion d'un client, le serveur doit lui renvoyer son numéro de connexion (défini par un compteur géré et maintenu par le serveur). De son côté le client reçoit son numéro de connexion et l'affiche sur une console.
Donnez les programmes associés au serveur et au client. On considèrera que les processus serveur et clients sont exécutés sur la même machine physique et que le serveur peut être contacté sur le numéro de port 55555.

**EXERCICE 8 : Communication en mode datagramme ou en mode connecté**
Des client $C_i$ envoient des messages à deux serveurs S1 et S2, S1 et S2 étant sur une même machine MS, S1 sur le port PORT1 (par exemple de numéro 55558) et S2 sur le port PORT2 (par exemple de numéro 55559). Les messages sont typés et composés :

soit d'un opérande x
soit de deux opérandes x et y