TP n°4 : Processus et parallélisme

CORRECTION

Exercice 1 – Visualisation de processus

Solution :

1) `ps` permet d'obtenir la liste des processus qui tournent au moment où on lance la commande, c'est pour cette raison que les processus utilisateur `bash` (l'interpréteur de commandes du terminal) et `ps` (la commande qui vient d'être lancée) apparaissent. Par contre, les processus `firefox` et `gedit` n'apparaissent pas car ce ne sont pas des processus utilisateur mais des processus système. Si l'on souhaite lister tous les processus, il faut taper la commande `ps -e` ou `ps -ef`. La colonne `UID` (*User ID*) indique le nom de l'utilisateur qui a lancé la commande.

2) La fenêtre du navigateur `firefox` se ferme. La fenêtre du terminal se ferme car elle correspond à l'interpréteur `bash` que nous venons de tuer.

3) Il n'est pas possible de lancer d'autres commandes car firefox a été lancé en avant-plan. L'appui CTRL-C ferme la fenêtre firefox en indiquant que le processus a été arrêté.

Exercice 2 – La fonction fork()

Solution : Voir corrigé du TD.

Exercice 3 – La fonction execl()

Solution :

1)
```
#include <stdio.h>

int main(int argc, char * argv[]) {
      if(argc != 2)
            printf("usage: affichez message\n");
      else
            printf("%s\n", argv[1]);
      return 0;
}
```

2)
```
#include <unistd.h> /* necessaire pour les fonctions exec */
#include <sys/types.h> /* necessaire pour la fonction fork */
#include <unistd.h> /* necessaire pour la fonction fork */
#include <stdio.h> /* necessaire pour la fonction perror */
```

```
int main() {
pid_t pid;
if ((pid = fork()) < 0)
      perror("fork error");
else if (pid == 0) {
            if (execl("/comptes/queudet-f/affichez","affichez","salut",
      (char *) 0) < 0)
                  perror("execl error");
      }
}
return 0;
}
```

Exercice 4 – La fonction kill()

Solution :

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
int main() {
int i=0;
int pidfils=fork();
if(pidfils!=0)
{
      sleep(10);
      kill(pidfils,SIGKILL);
} else
{
      while(1) {
            sleep(1); i++;
            printf("%d \n",i);
      }
}
}
```

Exercice 5 – La fonction wait()

Solution :

```
#include <unistd.h> /* necessaire pour les fonctions exec */
#include <sys/types.h> /* necessaire pour la fonction fork */
#include <unistd.h> /* necessaire pour la fonction fork */
#include <stdio.h> /* necessaire pour la fonction perror */

int main(int argc, char * argv[]) {

pid_t pid1,pid2,pid_premier;
int status;

switch(pid1=fork()) {
      case -1:    perror("fork error");
                  break;
      case 0:     execlp("ls","ls",(char *) 0);
                  break;
```

```
      default:    switch(pid2=fork()) {
                        case -1:    perror("fork error");
                                    break;
                        case 0:     execlp("ps","ps",(char *) 0);
                                    break;
                        default:    break;
                  }
                  break;
      }
pid_premier = wait(&status);
wait(&status);
if (pid_premier==pid1)
{
      printf("Premier processus a finir : %d\n", pid1);
}
else
{
      printf("Premier processus a finir : %d\n", pid2);
}
return 0;
}
```