Compte-rendu TP1 OS302

Dougy Hugo & Creton Pierre

Exercice 1:

Le programme créer un processus fils pid_fils. On regarde si on est dans le processus père ou fils. Dans le cas, où on est dans le processus fils, on affiche le pid du processus fils, et celui du père. Ensuite, le processus se temporise pendant la période `SLEEP_TIME`. Il affiche ensuite qu'il s'est réveillé et envoie le signal `EXIT_SUCCESS`.

Dans le cas, où on est dans le processus père, on affiche le pid du père puis celui du fils. On utilise ensuite la fonction wait pour attendre la fin du processus fils. Une fois que le processus fils est terminé, on affiche son pid avec le retour de la fonction wait. Pour finir, il affiche la valeur de l'état du processus fils. Il affiche les deux octets de etat, ce qui donne 0000 car il y eu un fonctionnement normal.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$ ./sync
Je suis le processus pere de pid 6353
*** PERE ***
Processus pere de pid 6353
Fils de pid 6354
J'attends la fin de mon fils...
*** FILS ***
Processus fils de pid 6354
Pere de pid 6353
Je vais dormir 30 secondes ...
Je me reveille
Je termine mon execution par un `return EXIT_SUCCESS`
Mon fils de pid 6354 est termine
Son etat etait : 0000
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$
```

Exercice 2:

Pour ce programme, on utilise la fonction `execl` pour exécuter un programme externe. On passe un argument à la fonction qui est le programme à exécuter. Ce programme fonctionne comme celui précédemment, sauf qu'à l'exécution du fils, on exécute le programme donné en argument. On regarde d'abord si la fonction a bien un argument, et on teste si l'exécution de execl a bien fonctionné.

On peut voir avec l'état la valeur que l'on a décidé de retourner dans le programme que l'on appelle avec `execl`, sur le screen on voit que l'on avait un return 92 (0x5c) à la fin du programme appelé.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$ ./fexec exec
Je suis le processus 7212 je vais faire fork
Pere numero 7212 attend
Coucou ! je suis le fils 7213
7213 : Code  remplace par exec
Coucou, ici 7213 !
Le fils etait : 7213 ... son etat etait :5c00 (hexa)
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$
```

Exercice 3:

Question 3

Un utilisateur réel est celui qui exécute le processus sur la machine. Un utilisateur effectif est l'identité que l'on prend (celle d'un autre utilisateur) lorsque c'est nécessaire pour exécuter un programme et donc de prendre les droits dont il dispose (mais seulement pour l'exécution de ce programme).

Question 4

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

int main (){
    struct passwd *user;
    struct group *group;

    uid_t uid = getuid();
    gid_t gid = getgid();
    uid_t euid = geteuid();
    gid_t egid = getegid();

    user = getpwuid(uid);
//on crée user comme struct passwd et on utilise la fonction getpwuid à qui on donne en argument l'uid de l'utilisateur réel pour pouvoir récupérer son nom

    group = getgrgid(gid);
//on crée group comme struct group et on utilise la fonction getgrid à qui on donne en argument le gid de l'utilisateur réel pour pouvoir récupérer son nom de groupe

```

```
22      printf("Reel : %s (UID = %d) \t %s (GID = %d)\n",user->pw_name,uid,
            group->gr_name,gid);
23  //on affiche le nom et l'uid de l'utilisateur réel ainsi que le nom et
        le gid de son groupe
24
25
26      user = getpwuid(euid);
27  //on modifie user pour pouvoir récupérer cette fois le nom de l'
        utilisateur effectif
28      group = getgrgid(egid);
29  //de même avec group pour pouvoir récupérer le nom de groupe effectif
30
31      printf("Effectif : %s (UID = %d) \t %s (GID = %d)\n", user->pw_name
            , geteuid(), group->gr_name, getegid());
32
33      //on affiche le nom et l'euid de l'utilisateur effectif ainsi que
            le nom et le egid de son groupe
34    return 0;
35  }
```

Question 5-6

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$ ./user
Reel : hugo (UID = 1000)         hugo (GID = 1000)
Effectif : hugo (UID = 1000)     hugo (GID = 1000)
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$ ls -l user*
-rwxr-xr-x 1 hugo hugo 16928 mars  11 18:30 user
-rw-r--r-- 1 hugo hugo   560 mars  11 18:29 user.c
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$ ls -ln user*
-rwxr-xr-x 1 1000 1000 16928 mars  11 18:30 user
-rw-r--r-- 1 1000 1000   560 mars  11 18:29 user.c
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP1_partie_obligatoire$
```

On peut voir que le résultat attendu est correct. On a bien le nom de l'utilisateur réel et le nom de l'utilisateur effectif qui a créé le programme ainsi que son UID et son EUID, il en est de même pour le groupe.

Question 7

```
root@debian:/home/hugo/Téléchargements# ls -ln users*
-rwsr-sr-x 1    0    0 16928 mars  11 19:14 users
-rw-r--r-- 1 1000 1000   518 mars  11 19:14 users.c
root@debian:/home/hugo/Téléchargements# ./users
Reel    : root(UID=0), root(GID=0)
Effectif        : root(UID=0), root(GID=0)
root@debian:/home/hugo/Téléchargements# déconnexion
hugo@debian:~/Téléchargements$ ./users
Reel    : hugo(UID=1000), hugo(GID=1000)
Effectif        : root(UID=0), root(GID=0)
hugo@debian:~/Téléchargements$
```

Exercice 4:

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


void fils(int i);
void pere();
/*
 * Dans main, on indique comment  utiliser les parametres passes sur la
 * ligne de commande
 */
int main(int argc, char *argv[])
{
    int nb_proc;
    int i_fils;
    pid_t pid_fils;

    if (argc != 2) {
        printf("Utilisation : %s nbre-de-processus !\n", argv[0]);
        return EXIT_FAILURE;
    }
nb_proc = atoi(argv[1]); /* Retourne 0 si argv[1] n'est pas un nombre
   */
```

```
24
25      /* Creation des processus fils */
26      for (i_fils = 1 ; i_fils <= nb_proc ; i_fils++) {
27          pid_fils = fork();
28          switch (pid_fils) {
29          case 0:
30              fils(i_fils);       /* il faut ecrire la fonction fils ...
                    */
31              break;
32          case -1 :
33              perror("Le fork n'a pas reussi");
34              return EXIT_FAILURE;
35          }
36      }
37      /*Dans la fonction pere, on utilisera le fait que wait renvoie
38      la valeur -1 quand il n'y a plus de processus fils a attendre.
39      */
40      pere();
41
42      return EXIT_SUCCESS;
43  }
44  void fils(int i){
45      printf("Je suis le fils %d, mon pére est %d\n",getpid(),getppid());
46      sleep(2*i);
47      exit(i);
48  }
49  void pere(){
50      int etat;
51      pid_t ret_wait;
52  /*on va faire une boucle do while pour pouvoir attendre et recuperer l'
        etat
53  et le pid du fils qui s'arrete jusqu'a ce que tous les fils s'arretent
        */
54      do{
55          ret_wait=wait(&etat);
56          if(ret_wait != -1) printf ("Le fils : %d c'est termine et son
                etat etait : %04x\n",ret_wait,etat);
57      }while (ret_wait != -1);
58
59      printf("C'est la fin !\n");
60  }
```

Travail Collaboratif:

Durant ce TP, nous nous sommes réparti les taches afin d'être plus efficace.

Pour l'exercice 1 & 2: nous avons ensemble regardé le fonctionnement du code et expliqué son fonctionnement.

Pour l'exercice 3: Nous avons regardé les pages man pour comprendre les différentes fonctions utilisés. De plus, un membre du binome a cherché la différence entre l'utilisateur réel et effectif pendant que l'autre a fait les tests du programme users.

Pour l'exercice 4: Nous avons tout deux implémenté,commenté et testé le code.