TP2 - Clément Bois et Filipe Doutel Silva

1 - Manipulation des processus Unix

Aucune question

2 - Création des processus Unix

2.1 - Primitive fork()

2.1.1 Exemple 0

1: On voit que les deux threads sont executé l'un après l'autre, c'est du moins l'impression qu'on a aux premiers abord.

2: Si on augmente d'un facteur 10, on voit que les threads ne sont pas executés les uns après les autres

3: Non, à chaque lancement le résultat sera différent, car le scheduler ne va pas ordonnancer de la même façon les deux processus à chaque fois

4: Le plus juste serait de les executer dans l'ordre pendant une période égale de temps et la plus efficaces serait des lancer les plus "léger" au debut en premier puis les plus lourd à la fin

2.1 - Codage différencié

1: On voit que le père et le fils comptent, et que le scheduler les laisse compter chacun leurs tours

Le principe est le même que pour la 2.1.1-4 , si la boucle est trop courte et donc trop rapide à s'executer, ont va avoir l'impression que les processus sont executé les uns après les autres.

Si la boucle est plus longue à etre executé on se rend compte que les processus ne sont pas executé les uns après les autres, mais selon les "directives" du scheduler. Les deux processus ne vont pas pouvoir compter jusqu'au bout sans être intérompus.

3 - Terminaison des processus

3.1 - Processus orphelin

Il y a création de processus car le père n'attend pas le fils (la fonction wait permet de le faire).

C'est pour ça que le père peu finir avant le fils laissant sont fils orphelin.

Le père et le fils vivant

```
503 14473 13177   0  9:26PM ttys001    0:00.00 ./orphelin.o
503 14474 14473   0  9:26PM ttys001    0:00.00 ./orphelin.o
```

On voit que le nouveau père du processus c'est le processus avec le PID 1

```
503 14474     1   0  9:26PM ttys001    0:00.00 ./orphelin.o
```

3.2 - Processus zombie

Il y a création d'orphelin car le fils meurt sans que le père soit au courant, pour le père le fils est toujours vivant, d'ou le processus zombie, il ne peux pas être tué.

```
| |     \-+= 16389 filipe ./zombie.o
| |       \--- 16390 filipe (zombie.o)
```

Le processus zombie entre parenthèse et le processus zombie (sur linux les processus zombie on sur mac ils sont entre parenthèses

3.3 - Primitive wait()

```
/* Sans la commande kill */

 Le fils 18334 s ' est termine correctement : 0
```

```
/*Après la commande kill -9 18366*/
Le fils 18366 s ' est mal termine : 9
```

Avec ce code on obtient la sortie suivante avec pstree:

```
int exit_cond;
pid_t tabProc [5];

for (int i = 0; i < 5; ++i) {
        if ((tabProc[i] = fork()) == -1) {
            printf("Erreur lors du fork");
        } else if (tabProc[i] == 0) {
            printf(" Pid du fils = %d\n", getpid());
            sleep(10);
            exit(0);
        }
    }
```

```
 | |     \-+= 18746 filipe ./wait.o
 | |       |--- 18747 filipe ./wait.o
 | |       |--- 18748 filipe ./wait.o
 | |       |--- 18749 filipe ./wait.o
 | |       |--- 18750 filipe ./wait.o
 | |       \--- 18751 filipe ./wait.o
```

Avec un seul wait on obtient la sortie suivante (entre parenthèses sont les processus zombie, sur mac ces processus sont entre parenthèses sur linux, ils sont suivie de la mention defunct) :

```
 | |     \-+= 19149 filipe ./wait.o
 | |       |--- 19151 filipe (wait.o)
 | |       |--- 19152 filipe (wait.o)
 | |       |--- 19153 filipe (wait.o)
 | |       \--- 19154 filipe (wait.o)
```

On corrige le problème avec le code suivant:

```
for (int i = 0; i < 5; ++i) {
        tabProc[i] = wait(&exit_cond);
        if (WIFEXITED(exit_cond))
            printf(" Le fils %d s ' est termine correctement : %d\n ",
                   tabProc[i], WEXITSTATUS(exit_cond));
        else
            printf(" Le fils %d s ' est mal termine : %d\n", tabProc[i],
WTERMSIG(exit_cond));
    }
```

4 - Gestion asynchrone de la terminaison

4.1 Signal SIGCHLD

L'intérêt de SIGCHLD c'est que le processus père peut réaliser d'autre tâche sans être bloqué et d'attendre ces fils, il peut par exemple faire un sleep, être intérompu par un fils qui vient de mourir, et ensuite continuer à sleep (ou tout autre activité). Très utile pour le prochain TP, le shell.

Sortie du programme:

```
Pid du pere = 20105
 Pid du fils = 20106
 Pid du fils = 20107
 Attente de la terminaison du fils ...
 Le fils 20106 s ' est termine correctement : 0
  Attente de la terminaison du fils ...
 Le fils 20107 s ' est termine correctement : 0
  Mort du pere !
```