UNIVERSITÉ CAEN NORMANDIE

TD/TP : Les processus

22.10.2018

Introduction

Dans ce TD/TP, nous allons voir comment se comportent les processus qu'ils soient "**normaux**", ***daemon*** ou ***zombie*** et comment on peut les créer, les lancer, les tuer etc...

Le but n'est pas de savoir écrire en langage C ! ! ! Je vous invite à vous appuyer sur le support de cours ! ! !

Si vous n'avez pas le temps de tout faire en TD/TP, rien ne vous empêche de continuer chez vous ☺

Rappel :

```
man ps
[...]
PROCESS STATE CODES
Here are the different values that the s, stat and state output specifiers
(header "STAT" or "S") will display to describe the state of a process.
D    Uninterruptible sleep (usually IO)
R    Running or runnable (on run queue)
S    Interruptible sleep (waiting for an event to complete)
T    Stopped, either by a job control signal or because it is being traced.
W    paging (not valid since the 2.6.xx kernel)
X    dead (should never be seen)
Z    Defunct ("zombie") process, terminated but not reaped by its parent.
```

1 Exercice 1

Soit le programme en C suivant : ***plein_processus.c***

```
1 #include <unistd.h>
  int main(){
  fork(); fork(); fork();fork(); fork(); fork();
  sleep(300);
  return 0;
6 }
```

1. **Donnez** le nombre de processus **plein_processus** créés et dormant 300s, **expliquez** le mécanisme et **dessinez** un graphe en fonction du temps correspondant à l'exécution de ce programme
2. **Lancez** les commandes ***ps -jf*** et ***pstree -p*** qui renvoient respectivement :

```
root@debian95-Rx-Sys-Fougeray:~/TD-TP-Systeme/TD_TP/processus# ps jf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
  994  1951  1951  1951 pts/1     1951 Ss+      0   0:00 -bash
  994  1000  1000  1000 pts/0     2202 Ss       0   0:00 -bash
 1000  2194  2194  1000 pts/0     2202 S        0   0:00  \_ ./plein_processus
 2194  2195  2194  1000 pts/0     2202 S        0   0:00  |   \_ ./plein_processus
 2195  2199  2194  1000 pts/0     2202 S        0   0:00  |   |   \_ ./plein_processus
 2199  2201  2194  1000 pts/0     2202 S        0   0:00  |   |   |   \_ ./plein_processus
 2195  2200  2194  1000 pts/0     2202 S        0   0:00  |   |   \_ ./plein_processus
 2194  2196  2194  1000 pts/0     2202 S        0   0:00  |   \_ ./plein_processus
 2196  2198  2194  1000 pts/0     2202 S        0   0:00  |   |   \_ ./plein_processus
 2194  2197  2194  1000 pts/0     2202 S        0   0:00  |   \_ ./plein_processus
```

```
-bash(1000)---plein_processus(2194)---plein_processus(2195)---plein_processus(2199)---plein_processus(2201)
                                                            `-plein_processus(2200)
                                    |-plein_processus(2196)---plein_processus(2198)
                                    `-plein_processus(2197)
```

3. **Donnez** le nombre de processus créés et dormants 30s si on avait 10 ***fork***.
4. **Donnez** la valeur retournée par la commande : ***ps aux | grep plein_processus | wc -l***
5. **Expliquez** ce qu'il faudrait faire pour n'avoir que 10 processus. **Écrivez** en langage humain le code.
6. **Écrivez** un programme dans lequel un père crée 3 fils qui dorment chacun pendant 10s.
   Le père affiche l'arborescence des processus (commande ***ps jf***).
   Vous utilisez pour cela les primitives ***fork***, ***system*** et ***sleep***.

2 Exercice 2 : de Zombie à Orphelin !

1. **Saisissez** le code suivant **zombie.c**, **Compilez** le, **Lancez** le et **lancez** la commande ***ps jf***

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char **argv){
  if (fork() == (pid_t)0){
    printf("fin du fils de %d\n",getpid());
        // sleep(30);
    exit(0);}
  // On est dans le père qui dort 30s
  sleep(30);
  printf("le père se réveille et stoppe : %d\n",getpid());
  return 0;}
```

2. **Interprétez** le résultat et **justifiez** le.
3. **Mettez** en commentaires la ligne ***sleep(30);*** du père et **ôtez** le commentaire de celle du fils.
4. **Renommez** le source de ***zombie.c*** en ***orphelin.c*** **Compilez** le, **Lancez** le et **lancez** la commande ***ps jf***
5. **Interprétez** le résultat et **justifiez** le.
6. **Écrivez** le programme, composé des 2 programmes du cours et permettant de **synchroniser** le père avec le fils.
   Voir les parties de cours
   — 4 : **créer un processus**
   — 6 : **Le père attend son fils**.
   Le père doit attendre la fin de son fils avant de se terminer.

3 Exercice 3 : Un daemon

1. **Récupérez** le programme **daemon** sur le site http://www.enderunix.org/documents/eng/daemon.phpet **appelez** le ***notre-daemon.c***
2. **Modifiez** le de manière à ne plus avoir de warning (voir le support de cous ! ! !)
3. **Lancez** le sans **l'esperluette**, **expliquez** ce qui se passe.
4. **Lancez** la commande ***ps | grep notre-daemon*** et **interprétez** le résultat
5. **Lancez** le à nouveau sans l'esperluette, **lancez** la commande ***ps | grep notre-daemon*** et **interprétez** le résultat que vous **justifiez**

4 Exercice 4 : synchronisation

L'objectif de cet exercice est d'écrire un programme dont le résultat est ce qu'il y a dans l'encadré.

1. **Donnez** le nombre de processus à créer.
2. **Expliquez** pourquoi on peut les nommer fils, père, grand-père et arrière grand-père.
3. **Écrivez** en langage "humain" l'algorithme de ce programme.
4. **Rappelez** le principe de la primitive **Waitpid()**, et **donnez** un exemple
5. Votre programme devra aussi effacer la console au démarrage, **donnez** la ligne de code le permettant.

fils : je suis le processus 7441 mon père à pour pid : 7440
fils : moi je dors 5 secondes pendant que les autres attendent...

père : je suis le processus 7440 mon père à pour pid : 7439
père : j'attends la fin de 7441

grand père : je suis le processus 7439 mon père à pour pid : 7435
grand père : j'attends la fin de 7440

arrière grand père : je suis le processus 7435
arrière grand père : j'attends la fin de 7439

fils j'ai fini de dormir pendant 5s enfin
père : je termine enfin
grand père : je termine enfin
arrière grand père : je termine enfin

5 Exercice 5 : Zombie le retour

Soit le code suivant : ***zombie2.c***

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char *argv[]) {
        int pid;
        int r;
        if ((pid=fork()) == 0){
                // Fils
                sleep(1);
                printf("\tfils %d de %d\n", getpid(), getppid());
                sleep(30);
                printf("\tmort fils %d de %d\n", getpid(), getppid());
                exit(0);}
        // Père
        printf("\tpère %d de %d\n", getpid(), pid);
        sleep(100);
        wait(&r);
        printf("\tpère %d de %d mort (%d)\n", getpid(), pid, r);
        sleep(100);
        return 0;}
```

Expliquez ce qui se passe après l'avoir lancé et **usé** de la commande ***ps -jf***

Réponse : A la création il y a un père et son fils. Le fils vit 30 secondes avant de mourir donc pendant 30 secondes, en utilisant intelligemment ***ps -edfl | grep zombie2*** tu les vois tous les deux. Ensuite le fils meurt mais le père continue à attendre 70 secondes durant lesquelles le ps montrera encore le pid du fils à l'état de defunct/zombie.

Ensuite le père se réveille en se demandant où est son fils et là, le ***district attorney*** lui dit qu'il est crevé alors il pleure encore 100 secondes durant lesquelles le ps ne te montre plus que lui. Puis il finit sa vie lamentable.

Provient de : https://www.developpez.net/forums/d1545567/c-cpp/c/fork-processus-zombie-processus-orphelin/

Concluez

1. Sur la création des processus et comment faire que le père n'ai pas trop d'enfants...
2. Sur les types de processus et leurs états
3. Sur la synchronisation.