Initiation à la Programmation des Processus sous Linux

(création des processus)

Amine RAHMANI
Université d'Alger 1 –
Benyoucef Benkhedda

Conseil sur les langages de programmation :

Pour maitriser un langage de programmation facilement, il faut apprendre les 5 notions principales:

- Syntaxe générale du programme
- Déclaration des variables
- Déclaration des boucles
- Utilisation des structures de conditions
- Déclaration des procédures et fonctions

Le reste ce n’est qu’une question de bibliothèques et API

Identifiant de processus :

Chaque processus est connu par un identifiant unique à lui appelé « PID »

Chaque processus, sauf le processus « init » possède un processus père dans son PID est une information dans le processus fils appelée « PPID »

Le PID et PPID sont des numéros codés chacun en 16bits

Vous pouvez les voir par la commande « **ps** » ou « **top** »

Dans les langages de programmation C et python, les PID peuvent être récupérés par les fonctions « **getpid()** » et « **getppid()** »

Identifiant de processus :

C:

```c
#include <stdio.h>
#include <unistd.h>

int main ()
{
  printf ("L'identifiant du processus est %d\n", (int) getpid ());

  printf ("L'identifiant du processus parent est %d\n", (int) getppid ());

  return 0;

}
```

Python:

```python
import os


print "L'identifiant du processus est %d\n", os.getpid()


Print "L'identifiant du processus parent est %d\n",


os.getppid()
```

Création d’un processus :

1. La méthode « SYSTEM »:

Il est possible de créer un processus en appelant une commande système dans un code source C ou python

La méthode system est utilisée pour cet objectif

Création d'un processus :

C:

```c
#include <stdio.h>
#include <unistd.h>

int main ()
    {
            int return_value;
            return_value = system ("ls -l /");
            return return_value;

     }
```

Python:

```python
import os

os.system("ls -l /")
```

La fonction *system* renvoie le code de sortie de la commande shell. Si le shell lui-même ne peut pas être lancé, *system* renvoie *127*; si une autre erreur survient, *system* renvoie *-1*.

Création d’un processus :

2. La méthode « fork »:

La méthode fork permet de dupliquer un processus en créant un **processus fils** similaire à son **processus père**

Les processus continuent leurs exécutions du même endroit. Là où la fonction **fork** est appelée

Création d'un processus :

C:

```
#include <stdio.h>
#include <unistd.h>
int main ()
{
   pid_t child_pid;
   printf ("PID de processus du programme principal :
            %d\n",(int)getpid ());
   child_pid = fork ();
   if (child_pid != 0) {
       printf ("je suis le processus parent, ID : %d\n", (int)
            getpid ());
       printf ("Identifiant du processus fils : %d\n", (int)
            child_pid);
  }else
      printf ("je suis le processus fils, ID : %d\n", (int) getpid
            ());
  return 0;
}
```

Python:

```
import os


def parent():
   newpid = os.fork()
   if newpid == 0:
     print("process has no child \n")
   else:
     pids = (os.getpid(), newpid)
     print("parent: %d, child: %d\n" % pids)



parent()
```

Création d’un processus :

3. La famille des méthodes « exec »:

Les fonctions **exec** remplacent le programme en cours d'exécution dans un processus par un autre programme.

Lorsqu'un programme appelle la fonction **exec**, le processus cesse immédiatement d'exécuter ce programme et commence l'exécution d'un autre depuis le début, en supposant que l'appel à **exec** se déroule correctement.

Plusieurs fonctions **exec**:

Création d’un processus :

3. La famille des méthodes « exec »:

- Les fonctions qui contiennent la lettre **p** dans leur nom (**execvp** et **execlp**) reçoivent un nom de programme qu'elles recherchent dans le path courant; il est nécessaire de passer le chemin d'accès complet du programme aux fonctions qui ne contiennent pas de p.

- Les fonctions contenant la lettre **v** dans leur nom (**execv**, **execvp** et **execve**) reçoivent une liste d'arguments à passer au nouveau programme sous forme d'un tableau de pointeurs vers des chaînes terminé par NULL.

- Les fonctions contenant la lettre **l** (**execl**, **execlp** et **execle**) reçoivent la liste d'arguments via le mécanisme du nombre d'arguments variables du langage C.

- Les fonctions qui contiennent la lettre **e** dans leur nom (**execve** et **execle**) prennent un argument supplémentaire, un tableau de variables d'environnement. L'argument doit être un tableau de pointeurs vers des chaînes terminé par NULL. Chaque chaîne doit être de la forme "VARIABLE=valeur".

Création d'un processus :

C:

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
#include  <unistd.h>
int spawn (char* program, char** arg_list)
{
  pid_t child_pid;
    child_pid = fork ();
  if (child_pid != 0)
        return child_pid;
  else {
        execvp (program, arg_list);
    fprintf (stderr, "une erreur est survenue au sein de
execvp\n");
    abort ();
  }
}
```

```
int main ()
{
   char* arg_list[] = {
    "ls",
    "-l",
    "/",
    NULL
  };
  spawn ("ls", arg_list);
  printf ("Fin du programme principal\n");
  return 0;
}
```

Création d'un processus :

Python:

```
def parent():
    newpid = os.fork()
    if newpid != 0:
      pids = (os.getpid(), newpid)
      print("parent: %d, child: %d\n" % pids)
    else:
      args=["ls",
     "-l",
     "/",
     NULL
                ]
                os.execvp("ls", arg)
parent()
```