Université Côte d'Azur
Faculté des Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TD n ° 2 : Processus en Unix, synchronisation père-fils

1 Relations entre processus père et fils

Un processus termine son exécution en appelant la primitive `sys.exit(statut)`. La valeur de `statut` est utilisée pour renvoyer un code de retour qui donne des informations sur la terminaison du processus. Habituellement, on utilise la valeur 0 pour une terminaison normale, et une valeur différente de 0 pour signaler une condition anormale (cette valeur indiquant la nature de l'anomalie selon une convention fixée).

Lorsqu'un processus se termine, il ne restitue pas toutes ses ressources et reste dans un état appelé "zombie", tant que son père n'a pas pris connaissance de son statut, par une primitive `os.wait` ou `os.waitpid`. Ces primitives sont utilisées par un processus père pour attendre la fin d'un ou plusieurs de ses fils [1].

Nous utilisons la primitive `os.waitpid`, définie comme suit :

```
import os
pid, statut = os.waitpid(pid, options)
```

Le paramètre `pid` permet de sélectionner le processus dont on attend la fin [2].

— si `pid` > 0, attendre la fin du processus fils de numéro `pid`

— si `pid` $= -1$, attendre la fin d'un processus fils quelconque

La valeur renvoyée est un tuple contenant le numéro du processus fils (zombie) effectivement pris en compte et l'information sur l'état du processus terminé. Pour l'interpréter, il est préférable d'utiliser des fonctions prédéfinies, par exemple :

— `os.WIFEXITED(statut)` vrai si le fils s'est terminé normalement (non interrompu par un signal), faux sinon.

— `os.WEXITSTATUS(statut)` donne le statut de sortie (code de retour) du fils (uniquement si la primitive précédente a renvoyé vrai)

Le programme ci-après lance un processus qui attend la fin de ses fils et imprime leur code de retour.

```
import errno, os, sys

N_wait = 20
for i in range(N_wait):
```

1. si un processus se termine sans avoir attendu la fin de ses fils, ceux-ci sont rattachés au processus de numéro `pid` $= 1$, qui finira par les éliminer.
2. il y a d'autres possibilités liées aux groupes de processus, non examinées ici.

```
    pid = os.fork()
    if pid == 0:  # child
        sys.exit(100 + i)
try: # parent waits for all of its children to terminate
    pid, status = os.waitpid(-1, 0)
    while True:
        if os.WIFEXITED(status):
            print("child {} terminated normally with exit status={}".\
            format(pid, os.WEXITSTATUS(status)))
        else:
            print("child {} terminated abnormally".format(pid))
        pid, status = os.waitpid(-1, 0)
except OSError as e:
    print("waitpid error: {}, {}".format(errno.errorcode[e.errno], os.strerror(e.errno)))
    if e.errno == errno.ECHILD:
        print("No more children left. Bye")
sys.exit(0)
```

1.1 Question 1

Modifier ce programme pour qu'il affiche les numéro des processus dans l'ordre dans lequel ils ont été créés.

1.2 Question 2

On considère le programme suivant :

```
import os, sys

print("Hello")
pid = os.fork()
print("ici: {}".format(pid))
if pid != 0:
    pid_wait, status = os.waitpid(-1, 0)
    if pid_wait > 0:
        if os.WIFEXITED(status) != 0:
            print("là: {}".format(os.WEXITSTATUS(status)))
    print("Bye")
    sys.exit(2)
sys.exit(0)
```

Combien de lignes ce programme imprime-t-il ? Discuter les ordres possibles dans lesquels ces lignes sont imprimées.

2 Terminaison - à vous de jouer

1. Écrire un programme Python qui affiche son PID, puis crée un fils. Le père affiche son PID, puis affiche un message signalant que son fils est mort lorsque c'est bien le cas. Le fils affiche son PID, s'endort 5 secondes puis se termine.
2. Écrire un programme Python qui affiche son PID, puis crée un fils. Le père affiche son PID, attend la terminaison du fils puis affiche son code de sortie.