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

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TD n ° 1 : Création de processus

L'objectif de ce TD [1] est d'approfondir les notions relatives aux processus et de les appliquer au cadre spécifique d'Unix. Ces notions seront appliquées dans le TP1. Ce texte ne donne pas tous les détails sur les primitives décrites. En cas de besoin, pour les TP, utiliser l'aide de Python (modules `os, time, sys`).

La primitive `fork()` crée un processus "fils" du processus appelant (le "père"), avec le même programme que ce dernier. La valeur renvoyée par `fork()` est :

— au père : le numéro (PID) du processus fils.

— au fils : 0.

En cas d'échec (table des processus pleine), aucun processus n'est créé, et l'exception `OSError` est levée.

1 Question 1

Qu'affiche l'exécution du programme suivant :

```
import os, sys

x = 1
pid = os.fork()
if pid == 0: # child
    x = x + 1
    print("child: x = ", x)
    sys.exit(0)
# parent
x = x - 1
print("parent: x = ", x)
sys.exit(0)
```

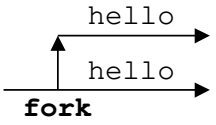
2 Question 2

On considère les deux programmes suivants et le schéma de leur exécution (l'axe du temps est orienté vers la droite).

1. Plusieurs des figures et exemples sont empruntés à R. E. Bryant, D. O'Hallaron. *Computer Systems : a Programmer's Perspective*, Prentice Hall, 2003.

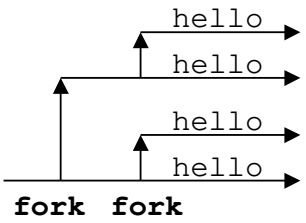
```
import os, sys

os.fork()
print("hello!")
sys.exit(0)
```




```
import os, sys

os.fork()
os.fork()
print("hello!")
sys.exit(0)
```



Illustrer l'exécution du programme obtenu en ajoutant un troisième `fork()`.

3 Question 3

Combien de lignes "hello !" affiche chacun des deux programmes suivants ?

```
import os, sys

for i in range(2):
    os.fork()
print("hello!")
sys.exit(0)
```

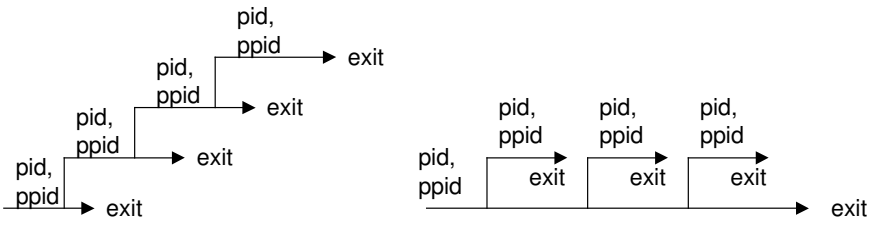
```
import os, sys

def do_it():
    os.fork()
    os.fork()
    print("hello")

if __name__ == "__main__":
    do_it()
    print("hello")
    sys.exit(0)
```

4 Question 4

On considère les deux structures de filiation (chaîne et arbre) représentées ci-après.



Écrire un programme qui réalise une chaîne de `n` processus, où `n` est passée en paramètre de l'exécution de la commande (par exemple, n = 4 sur la figure ci-dessus). Faire imprimer le numéro de chaque processus et celui de son père. Même question avec la structure en arbre.

En python, on recupère les arguments du programme dans la *liste* `sys.argv`. Attention, le premier élément de la liste (`sys.argv[0]`) contient le nom du programme. Le premier argument passé au programme se trouve donc dans `sys.argv[1]`. Pour savoir combien d'arguments ont été donnés, il suffit d'utiliser la fonction `len()` :
`if len(sys.argv) == 1: ...`

5 Question 5

1. Écrire un programme Python qui affiche son PID, puis crée deux fils. Ils affichent leur PID et leur filiation et se terminent.
2. Écrire un programme Python qui affiche son PID, puis crée un fils et un petit-fils. Ils affichent leur PID et leur filiation et se terminent.
3. Écrire un programme Python qui change de PID toutes les 2 secondes, et affiche à chaque fois celui-ci.

   Note : `time.sleep(n)` endort un processus pendant `n` secondes. `n` est de type `float`.

6 Création de fils

1. Écrire un programme Python qui crée 5 fils. Le programme et chacun des fils affiche son PID et se termine.
2. Modifier le programme pour que le nombre de fils qu'il crée soit reçu en argument de la ligne de commande.
3. Modifier le programme pour que chaque fils affiche de plus son rang dans la fratrie.

7 Création de petits-fils

1. Écrire un programme Python qui crée 3 fils, chacun des fils créant 2 petits-fils. Le programme et chacun des fils ou petits-fils affiche son rang dans la fratrie, son PID et se termine.
2. Modifier le programme pour que le nombre de fils et de petits-fils qu'il crée soient reçus en argument de la ligne de commande.