PROGRAMMATION CONCURRENTE

CPE Lyon – 3ETI

TRAVAUX PRATIQUES SEANCE 1 – REVISION PROGRAMMATION PYTHON & fork() & wait()

Exercice 1 – Arguments de la ligne de commande – Chaînes de caractères

❶ La variable **`sys.argv`** contient les arguments de la ligne de commande, sous forme d'une liste dont le premier élément est le nom du programme lancé. La valeur **`sys.argv[0]`** est une chaine de caractères contenant le nom du script, **`sys.argv[1]`** est une chaine de caractères contenant le premier argument, … etc.

Recopiez le programme suivant et testez-le avec la ligne de commande suivante :

```
$ python3 exercice1.py python 3 ETI 2024
import sys
print("Nom du programme : ", sys.argv[0])
print("Nombre d'arguments : ", len(sys.argv)-1)
print("Les arguments sont : ")
for argument in sys.argv[1:] :
    print(argument)
```

Dans cet exemple, le programme **`exercice1.py`** est lancé avec **4** arguments.
Les **5** paramètres sont reçus à partir de la ligne de commande (le nom du programme + les arguments).

❷ Ecrivez un programme (**`miroir.py`**) qui prend en argument une chaîne de caractères et l'affiche à l'envers.

Par exemple :

```
$ python3 miroir.py trace
> ecart
```

❸ Modifiez le programme **`miroir.py`** (créez le fichier **`miroir2.py`**) pour qu'il traite plusieurs arguments.

Par exemple :

```
$ python3 miroir2.py ecart DNA saper
> trace AND repas
```

Exercice 2 – utilisation des arguments de la ligne de commande

Ecrire un programme (**`moyenne.py`**) qui calcule et affiche la moyenne d'un ensemble de notes (nombres entiers) passées en arguments sur la ligne de commande. Le résultat doit être affiché sous la forme :
La moyenne sera affichée tronquée à **2** décimales de précision.

Vérifier que :

- Au moins une note est passée en argument - Sinon, on affichera le message « **`Aucune moyenne à calculer`** ».

- Chaque note reçue en paramètre doit être un nombre entier dont la valeur est comprise entre **0** et **20** (bornes incluses). Si cette condition n'est pas respectée, on affichera le message : « **Note(s) non valide(s)** ».
- Si toutes les notes sont valides, on affichera la moyenne sous la forme : **`Moyenne = <valeur>`**
  <valeur> est la valeur de la moyenne.

Exemples :

$ python3 moyenne.py 10 15 15
`> Moyenne est : 13.33`

$ python3 moyenne.py 8 7 32 15
> **Note(s) non valide(s)**

$ python3 moyenne.py 10 3ETI
> **Note(s) non valide(s)**

$ python3 moyenne.py 4 8 4.5 7
> **Note(s) non valide(s)**

Note : le symbole **$** indique une entrée utilisateur sur la ligne de commande. Le symbole **>** indique un affichage du programme [*Ce ne sont pas des caractères à afficher*].

Aide

- Avant de développer le code, réfléchissez aux types des arguments de la ligne de commande. Quelle conversion est nécessaire ?
- Pour convertir une chaine de caractères représentant un nombre en un entier (ou un réel), vous pouvez utiliser les fonctions de conversion suivantes : **`int()`** ou **`float()`**.
- Pour afficher **2** décimales d'un nombre **`flottant x`**, vous pouvez utiliser la syntaxe suivante :

```
print("%.2f" %x)
```

Exercice 3 – *Initiation appel fork()*

Testez et commenter le programme suivant :

```
import os,sys
N = 10
v=1
while os.fork()==0 and v<=N :
      v += 1
print(v)
sys.exit(0)
```

Combien de « **`Bonjour`** ! » et de « **`Ok`** ! » affiche le programme suivant ? Testez et commenter ce programme.

```
for i in range(4) :
       pid = os.fork()
       if pid != 0 :
           print("Ok !")
       print("Bonjour !")
sys.exit(0)
```

Exercice 4 – *processus en cascade*

Écrire **deux programmes** qui créent **N** processus P_i tels que :

1. Dans le 1er programme : pour tout **`i`** entre **`2`** et **`N`**, **P_i** soit le fils de **P_{i-1}**
2. Dans le 2ème programme : pour tout **`i`** entre **`2`** et **`N`**, **P_i** soit le fils de **P_1**

Chaque processus devra afficher son identifiant (son **`PID`**) et celui de son père. Dessinez l'arbre des processus correspondant à l'exécution de chacun des 2 programmes.

Exemple d'affichage : **`Je suis le processus 1341, mon père est le processus 1339`**.

Exercice 5 – *Fork() & wait()*

Ecrire un programme qui crée deux processus fils, l'un affiche les entiers de 1 à 100 et l'autre de 101 à 200.

Modifier ce programme pour que les nombres s'affichent toujours dans l'ordre numérique croissant 1, 2, 3, ... , 200.

Exercice 6 – *Fork()*

Combien de fois le message « `3ETI` » sera-t-il affiché ?
Dessinez un diagramme de séquence indiquant quelles lignes sont exécutées par chaque processus. Pour vous aider, vous pouvez rajouter des **`print()`** pour afficher notamment le **`PID`** retourné par os.**`getpid()`**.

```
import os, sys
os.fork()
if (os.fork()) :
      os.fork()
print("3ETI 2024")
sys.exit(0)
```

Exercice 7 – *Fork()*

Dessinez l'arbre généalogique des processus engendrés par le programme ci-dessous. Qu'affiche ce programme ?

```
import os, sys
if (os.fork() != 0) :
        if (os.fork() == 0) :
                os.fork()
                print("1")
        else :
                print("2")
else :
        print("3")
print("4")
sys.exit(0)
```

Exercice 8 – *Fork() & wait()*

Ecrire un programme **`forkWait.py`** qui crée qui :

- Récupère en ligne de commande (sys.argv) un nombre **`N`** de processus qu'il doit créer.
- Une fois que les **`N`** processus fils sont créés, il se met en attente de la fin d'exécution de ses fils.
- Dès qu'un fils se termine, il affiche l'identité (PID) de ce fils et la valeur retournée par ce fils.

Chaque processus fils affiche son `PID` et le `PID` de son père, et ensuite il fait une pause [**`time.sleep()`**] de **`2*i`** secondes. Après la pause, il indique qu'il reprend son exécution avant de faire appel à **`exit()`** avec la valeur de `i` (**`i`** est le numéro de l'itération de la boucle de création des processus fils).