TD #5 – Processus

Préambule

Objectif : Utilisation basique des processus et premiers pas vers la parallélisation de tâches

Fichiers additionnels :

- example1.sh – script à appeler dans le premier exercice ; vous pouvez essayer de deviner ce que fait le programme en lisant la commande, puis vérifier à l'exécution

Appel à un processus extérieur (subprocessing)

En utilisant l'appel `subprocess.Popen()`, écrivez un programme qui appelle le script `example1.sh` et affiche sur le terminal ce que le script est censé afficher.

En avant les processus (multiprocessing)

Création simple de processus

En utilisant la structure `multiprocessing.Process`, écrivez les programmes correspondant aux comportements suivants :

- Affichage de "Hello World!" pour les deux processus ;
- Affichage de "Mon PID est ... et celui de mon pere/fils est ... !" ;
- Le processus fils choisit aléatoirement une valeur, l'affiche et la communique à son père qui l'affiche à son tour.

Refaites l'exercice en utilisant la fonction `os.fork()`.

Prêt à attendre

Écrivez un programme où un processus père créé 10 processus fils et attend qu'ils terminent. Chaque fils attend un nombre de secondes choisi aléatoirement entre 1 et 10, affiche son PID puis se termine. Le processus père affiche à chaque terminaison, le PID du processus fils qui a terminé son exécution.

Vous avez deux implémentations possibles pour cet exercice, choisissez celle que vous préférez, sachant que la seconde est plus simple :

1) En utilisant l'appel `multiprocessing.connection.wait()` : pour ceci, vous allez avoir besoin d'utiliser le sous-module `connection` de `multiprocessing` ; couplé avec le mécanisme de `Process.sentinel`, il permet d'attendre sur plusieurs événements à la fois, ici la fin de plusieurs processus.
2) En utilisant l'appel `os.wait()` à la place de `Process.join()` pour attendre n'importe quel processus.

Test de primalité

Écrivez un programme, nommé `prime-test.py` qui permet de paralléliser des tests de primalité ie. vérifier qu'un nombre est premier. Le programme prendra en entrée deux nombres :

- n, l'entier max à tester, sachant que l'on testera les entiers de 2 à n ;
- m, le nombre de processus à créer pour le partage de tâches.

Chaque test de primalité consiste en l'exécution de l'algorithme suivant :

```
i <- 2
tant que i <= sqrt(n) faire
    si n modulo i = 0 alors
        retourne FAUX
    i <- i + 1
retourne VRAI
```

Le programme doit produire l'affichage **ordonné** suivant :

```
$ ./prime-test.py 10 2
2: True
3: True
4: False
5: True
6: False
7: True
8: False
9: False
10: False
```