
GESTION DES PROCESSUS ET DES RESSOURCES

Exercice 0**Partie A**

1. Crée un script Python `infini.py` contenant une boucle infinie.
2. Ouvre un terminal et tape la commande `top` pour voir les processus exécutés en temps réel.
3. Ouvre un autre terminal et lance votre script Python avec la commande `python3 infini.py &` (il faut bien sûr être placé dans le repertoire de ce script).
4. Regarde le premier terminal pour voir le processus correspondant à l'exécution de ce script Python. Observe que ce processus monopolise l'ensemble des ressources du processeur.
5. Repère le PID de ce processus et stoppe son exécution avec la commande `kill <PIDduProcessus>`.

Partie B

1. Utilise la commande `cat /proc/cpuinfo` et repère le nombre de « processeurs » disponibles sur votre machine.
2. Ouvre un terminal et tape la commande `top` pour voir les processus exécutés en temps réel.
3. Crée un autre script Python appelé `factice.py` contenant le code suivant :

```

1 from timeit import timeit
2 def factice():
3     a = 0
4     for i in range(100000):
5         a = a**3
6     # la fonction timeit permet de lancer un certain nombre de fois une fonction
7     # et renvoie le temps total d'exécution temps = timeit(factice, number=100)
8     print(temps)

```

4. Exécute ce script dans un terminal via la commande `python3 factice.py` et note le temps d'exécution des 100 appels à la fonction `factice`.
5. Dans un autre terminal, tape la commande `python3 infini.py &` autant de fois qu'il y a de "processeurs" sur la machine. On va ainsi monopoliser l'ensemble des ressources processeurs de la machine avec des boucles infinies (cette fois-ci toutes les ressources!).
6. Dans le premier terminal, relance le script `factice.py`. Comment expliquer le ralentissement observé ?

Attention : N'oubliez pas de stopper les 4 processus infinis en exécutant `kill PID1 PID2 PID3 PID4` où `PIDx` sont les PID des 4 processus correspondants au script `infini.py`.

Exercice 1

Voici un script Python dans lequel on crée une variable globale `nombre` qui vaut 0 au départ et à laquelle on ajoute 100 quatre fois successivement grâce à la fonction `ajoute_100`.

```

1 import time
2 def ajoute_100():
3     global nombre
4     for i in range(100):
5         time.sleep(0.001) # pour simuler un traitement avec des calculs
6         nombre = nombre + 1
7 if __name__ == '__main__':
8     nombre = 0
9     for i in range(4):
10        | ajoute_100()
11        | print("valeur finale :", nombre)

```

1. Copie et exécute le script dans un terminal pour vérifier que la valeur finale de la variable `nombre` est bien égale à 400.

On va maintenant supposer qu'une telle variable est partagée par 4 processus, chacun étant chargé d'ajouter 100 à cette variable.