Threads, interblocage

Nous avons constaté dans le premier TP que lorsque des processus s'exécutent de manière concurrente, il est impossible de savoir à l'avance quel processus aura la main à un moment donné.

Nous avons également vu que lorsque des processus partagent des ressources exclusives, il peut y avoir interblocage.

Nous allons illustrer cela une dernière fois en créant des *threads* (processus légers).

PYTHON gère les threads très simplement, lorsqu'un processus qui est un programme Python crée des *threads*, ceux-ci peuvent accéder aux variables globales à l'aide de `global`.

Exercice 1

Ouvrir, lire, comprendre, puis lancer `thread1.py`.
Comparer la situation avec celle du premier TP : l'exécution des 3 *threads* est-elle déterministe ?

Exercice 2 : à exécuter sur www.online-python.com

On a expliqué que les threads, contrairement aux processus « lourds », partagent les variables globales.

1. Créer un script `thread2.py` qui comporte une variable globale `compteur`, initialisée à `0` et une fonction `f` qui
   - ne prend rien en entrée ;
   - ne renvoie rien mais à l'aide d'une boucle `for`, incrémente 1 000 000 fois `compteur`.
2. Créer 4 threads qui exécutent `f` en tâche de fond (ne pas définir la valeur de `args` lors de la création).
3. Lancer les threads (méthode `start`) et attendre qu'ils se terminent (avec la méthode `join`) puis afficher `compteur`.
4. Quelle devrait être la valeur de `compteur` ? Comment expliquer le phénomène ?

Exercice 3

Pour éviter le problème de l'exercice précédent, il faut s'assurer que pendant l'exécution

de `compteur += 1`, chaque thread ne sera pas interrompu.

Pour ce faire il suffit, dans le programme principal, de créer un « verrou » :
`verrou = threading.Lock()`
Ce verrou agit un peu comme une ressource exclusive : quand un thread acquiert le verrou, les autres threads qui cherchent à l'acquérir doivent attendre qu'il le libère.
Pour acquérir le verrou on utilise `verrou.acquire()`.
Pour le libérer on utilise `verrou.release()`.

1. Créer un fichier `thread3.py` et copier le contenu de `thread2.py` dedans.
2. Créer un verrou et modifier `f` pour que l'instruction `compteur += 1` ne soit pas interrompue.
3. Constater le résultat

Exercice 4

En définitive, les verrous apparaissant comme des ressources exclusives, on doit pouvoir simuler le phénomène d'interblocage du TP précédent (l'exercice portant sur le robot).

1. Ouvrir le fichier `thread4.py` et lire son contenu.
2. Compléter ce fichier.
3. Exécuter le programme plusieurs fois et commenter les résultats.