2023 – 2024

TP 3

1°) Reprendre le script vu en cours et le modifier de manière à lancer un nombre variable (> 2) de threads identiques avec chacun son propre temps de pause passé en paramètre.

2°) Créer un script Python multi-threadé qui demande un entier, lance un thread qui affiche l'entier suivant, lance un autre thread qui affiche son carré, et ce en boucle tant que l'entier saisi est différent de 0. Note : attendre la fin des threads secondaires avant de redemander un nouvel entier.

3°) La solution 2°) est gourmande en ressources (surtout compte tenu de la très faible complexité des tâches) car la création / destruction de thread n'est pas anodine. Écrire une solution qui lance les threads une seule fois et utilise une variable globale pour l'entier saisi. En somme :
- le thread principal demande un entier et le stocke dans une variable globale, en boucle tant que celui-ci est différent de 0
- un 1er thread : récupère l'entier stocké globalement et affiche l'entier suivant, et ce tant que l'entier récupéré est différent de 0
- un 2nd thread : récupère l'entier stocké globalement et affiche son carré, et ce tant que l'entier récupéré est différent de 0

Que se passe t-il ?

Rappel : les variables dans une fonction sont toujours locales sauf à ajouter l'instruction « global var », var étant le nom de la variable globale, avant toute utilisation dans la fonction

4°) La solution 3°) est encore pire que la 2°) car les threads secondaires ne sont certes créés qu'une seule fois mais ils tournent en boucle sans pause (« polling »), accaparant toutes les ressources. Il faut donc ajouter un moyen d'imposer une pause aux différents threads.
a) En remarquant qu'une demande de saisie clavier bloque un thread / processus, modifier 3°) de manière à marquer une pause dans tous les threads
b) Modifier 3°) pour mettre en œuvre une temporisation différente pour chaque thread secondaire (pause comprise entre 1 et 5 s). Observer les différences de comportement

Remarque : les threads secondaires n'ont rien à faire tant qu'on ne saisit pas une valeur au clavier. Il serait donc judicieux d'utiliser un mécanisme d'attente / synchronisation pour ordonnancer les différents threads. Pour cela, on peut utiliser un (des) événement(s) (cf cours).
On peut aussi utiliser, de manière détournée, des verrous. Une possibilité est d'utiliser 2 verrous, un par thread secondaire. Chaque verrou est verrouillé par le thread secondaire afin d'être bloqué après un traitement. Les verrous sont débloqués par le thread principal lorsqu'un nouveau nombre est saisi. A tester s'il vous reste du temps ...

5°) Écrire un script qui possède une chaîne comme variable globale et lance 5 threads secondaires identiques qui vont travailler sur cette chaîne. Elle est initialement vide.
Chaque thread secondaire :

- récupère la chaîne globale
- fait appel à un module comportant la méthode qui :
  - comporte 2 paramètres : un caractère ‘c’ et une chaîne ‘s’
  - ajoute (concatène) ‘c’ à la fin de ‘s’
  - attend une durée variable ‘d’ (fixée manuellement)
- le caractère ajouté est constant mais spécifique au thread (‘a’ pour le 1er thread par ex)
- affiche et stocke dans la variable globale le résultat de l’appel à la méthode décrite ci-dessus
- les threads s’arrêtent quand la chaîne fait 30 caractères

Le thread principal affiche la chaîne finale.
Note : len(ch) retourne la longueur (taille) de la chaîne « ch »

Que constatez-vous si d = 0 ? Et si d = 0,1 ?

6°) Identifier la section critique dans l’exercice 5°). Utiliser un mutex pour la protéger.
Dans un 1er temps, placer l’affichage en dehors de cette section critique.
Que constatez-vous si d = 0 ? Et si d = 0,1 ?
Maintenant, placer l’affichage dans la section critique. Que constatez-vous si d = 0 ? Et si d = 0,1 ?