# TP8 (3h) : Threads, synchronisation

*L’objectif de ce TP est de se familiariser avec les l’utilisation de threads. Nous allons utiliser le module* ***threading*** *de Python (cf.* [*https://docs.python.org/3.5/library/threading.html*](https://docs.python.org/3.5/library/threading.html) *).*

1. **Premiers pas avec les threads**

Récupérer le squelette du programme TP81.py sur webetud. Vous y trouverez une implémentation de l’algorithme de tri à bulles (très inefficace, mais utile ici car relativement simple).

Modifier le programme comme suit (*la réponse à* *chaque étape doit être rendue comme une version à part*) :

1. Créer un thread et lancer la procédure de tri sur ce thread. Le thread principal doit attendre la fin du tri avant d’afficher le résultat.
2. Créer une fonction **tri\_sable** qui marche de manière similaire que **tri\_bulle** mais fait le parcours en sens inverse : au lieu de remonter les éléments plus petits vers le début de la liste, elle fait descendre les éléments plus grands vers la fin. Tester cette fonction.
3. Le tri d’une liste devrait pouvoir se faire en appelant en parallèle les deux fonctions de tri, sur deux threads. Tester le résultat obtenu.  
   Pour s’assurer que le programme marche bien, écrivez une boucle qui vérifie que tous les éléments (0..N) se trouvent bien dans la liste après le tri. Que constatez-vous et comment expliquez-vous le résultat ?
4. Identifier les sections critiques des deux fonctions de tri et les rendre mutuellement exclusives afin de résoudre le problème précédent. Pour cela, ajouter un mutex (objet de type **threading.Lock**) ainsi que les synchronisations nécessaires à l’entrée / la sortie de ces sections critiques.
5. **Collecte de données de sondes de température**

**On demande de rendre un programme par question.**

**Attention : pour cet exercice il est nécessaire d’installer numpy pour python3**

**Pour cela dans la VM : sudo apt-get install python3-numpy**

Un programme appelé « superviseur » collecte des données de températures à partir de plusieurs capteurs. Le nombre de capteurs est fourni en argument à l’appel du programme « superviseur ».

Les capteurs de températures sont représentés par des threads dont le rôle est de générer un nombre aléatoire compris entre 0 et 32 toutes les secondes.

Le superviseur génère un thread par capteur. Les données collectées sont stockées dans une matrice de taille N \* 10 où N est le nombre de capteurs. La matrice sert à stocker les données collectées en 10 secondes. Les données correspondant à chaque capteur sont stockées sur une ligne correspondante de la matrice, qui sera gérée comme une *liste circulaire*.

En python, on utilisera le module **numpy** pour manipuler les tableaux et matrices. Pour créer un tableau m \* n initialisé avec des zéros on peut utiliser **zeros((m,n))**. Cela renvoie une variable de type **array**.

Sur une variable de type matrice de type **array** on peut extraire une ligne avec la syntaxe **matrice[indice\_ligne,:]**

**Q1**. Ecrire un programme implémentant un programme principal créant les threads des capteurs. Dans cette question, chaque capteur rentre les 10 valeurs de température des 10 secondes dans la matrice. Le programme principal attend que les capteurs aient terminé puis affiche les valeurs collectées.

**Q2**. Modifier le programme pour que les capteurs remplissent en continue la liste circulaire des données de température. Mettre en place une alarme qui toutes les 30 secondes permette au thread principal de créer un thread appelé “Stat\_Capteur” chargé de détecter toutes les 30 secondes la valeur maximale de la matrice et de l’afficher. Le thread “Stat\_Capteur” travaille sur une copie de la matrice.

Ainsi, on a :

-le thread principal “superviseur” créant les N threads de capteurs et le thread de Stat\_Capteurs (créé toutes les 30 secondes)

-les N threads de capteurs générant un nombre aléatoire entre 0 et 32 et stockant les données dans la ligne i de la matrice avec i compris entre 0 et N-1 ; les données sont gérées comme une liste circulaire.

-le thread de Stat\_Capteurs travaille sur une copie de la matrice afin que les Capteurs puissent potentiellement continuer à écrire les données dans la matrice.

Prévoir un mécanisme de protection pour que les threads qui lisent/écrivent la matrice en même temps n’interférent pas.

**Q3**. Trouver un moyen simple pour que tous les affichages à l’écran soient désormais dans un fichier “stats\_globales.txt”