# Compte rendu

# TP8

# Threads, synchronisation

# Ex. 1. Premiers pas avec les threads

a) **Principe :** On souhaite faire un programme capable de créer un thread dont le but et de lancer la procédure de tri à bulles sur une liste créée.

**Code source :**

```
def threadExec(x):#fonction qui sera exécuté par chaque thread, prend en paramétre la liste à trier
    tri_bulle(x)#fonction de tri suivant le tri à bulle
    print(x)#affichage de la liste après le tri
    pass

def tri_bulle(list):#fonction de tri à bulle, donnée dans la structure tp8
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(len(list)-2, -1, -1):
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i]
                list[i] = list[i+1]
                list[i+1] = temp

# création liste non-ordonnée pour les tests
N = 10
liste = []
for i in range(N):#on remplit la liste
    liste.insert(random.randint(0, i), i)
print(liste)#on affiche la liste non trié

t = threading.Thread(target=threadExec, kwargs={'x' : liste})#création du thread qui devra exécuter la fonctio
t.start()#lancement du thread
t.join()#le thread attend la fin de son exécution
```

**Vérification :**

```
[0]utilisateur@Ubuntu:~/Bureau$ python3 TP8Exo1.py
[6, 8, 1, 7, 0, 4, 2, 3, 5, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

b) **Principe :** On souhaite obtenir le même résultat que précédemment sauf que cette fois ci le on utilisera le tri sable.

**Code source :**

```python
#ce code est l'identique du précédent sauf que l'on réalise un tri à sable
def threadExec(x):
    tri_sable(x)
    print(x)
    pass

def tri_sable(list):
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(0, len(list)-1, 1):
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i+1]
                list[i+1] = list[i]
                list[i] = temp

# création liste non-ordonnée pour les tests
N = 10
liste = []
for i in range(N):
    liste.insert(random.randint(0, i), i)
print(liste)

t = threading.Thread(target=threadExec, kwargs={'x' : liste})
t.start()
t.join()
```

**Vérification :**

```
[0]utilisateur@Ubuntu:~/Bureau$ python3 TP8Exo1b.py
[6, 9, 5, 2, 0, 4, 3, 1, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

c) **Principe :** On souhaite créer deux threads qui vont parallèlement trier la liste, l'un avec tri bulle et l'autre avec tri sable. Le but était de constater que des décalages (pertes de valeurs) seront engendrées.

**Code source :**

```
def threadExecSable(x):
    tri_sable(x)
    print(x)
    pass

def threadExecBulle(x):
    tri_bulle(x)
    print(x)
    pass

def tri_sable(list):
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(0, len(list)-1, 1):
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i+1]
                list[i+1] = list[i]
                list[i] = temp

def tri_bulle(list):
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(len(list)-2, -1, -1):
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i]
                list[i] = list[i+1]
                list[i+1] = temp

# création liste non-ordonnée pour les tests
N = 1000
liste = []
for i in range(N):
    liste.insert(random.randint(0, i), i)
print(liste)
#cette fois on crée deux threads qui vont exécuter le tri à bulle et le tri à sable
t = threading.Thread(target=threadExecSable, kwargs={'x' : liste})
t2 = threading.Thread(target=threadExecBulle, kwargs={'x' : liste})
t.start()
t2.start()
t.join()
t2.join()
#On verifie si il y a des erreurs, cad si il y a des valeurs qui auraient disparu ou se seraient ma
for i in range(0,1000):
    if(i!=liste[i]):
        print("erreur")
        print(liste[i])
        print(i)
        break
```

**Vérification :**

```
934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949,
950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965,
966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981,
982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997,
998, 999]
erreur
349
348
```

d) **Principe :** On souhaite corriger ce problème de décalage, pour ce faire il faut empêcher les deux threads d'avoir accès à la liste en même temps, ou plutôt d'avoir de la trier en même temps et de se mélanger les valeurs. Pour cela, on va utiliser un mutex (objet de type threading.lock)

**Code source :**

```
#Ce code correspond au précédent sauf que cette fois ci on va s'assurer qu'il ne peut plus y av
def threadExecSable(x):
    tri_sable(x)
    print(x)
    pass

def threadExecBulle(x):
    tri_bulle(x)
    print(x)
    pass

def tri_sable(list):
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(0, len(list)-1, 1):
            lock.acquire()#le thread va attendre de pouvoir prendre un jeton pour réaliser cett
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i+1]
                list[i+1] = list[i]
                list[i] = temp
            lock.release()#une fois réalisée, il va reposer le jeton pour que le thread suivant

def tri_bulle(list):
    if len(list) <= 1:
        return
    temp = 0
    while temp != None:
        temp = None
        for i in range(len(list)-2, -1, -1):
            lock.acquire()
            if list[i] > list[i+1]:
                # permutation => temp != None
                temp = list[i]
                list[i] = list[i+1]
                list[i+1] = temp
            lock.release()

lock = threading.Lock()#création du lock, il contient un jeton et permettra déviter des erreurs
# création liste non-ordonnée pour les tests
N = 1000
liste = []
for i in range(N):
    liste.insert(random.randint(0, i), i)
print(liste)

t = threading.Thread(target=threadExecSable, kwargs={'x' : liste})
t2 = threading.Thread(target=threadExecBulle, kwargs={'x' : liste})
t.start()
t2.start()
t.join()
t2.join()

for i in range(0,1000):
    if(i!=liste[i]):
        print("erreur")
        print(liste[i])
        print(i)
```

**Vérification :**

```
 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965,
 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981,
 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997,
 998, 999]
[0]utilisateur@Ubuntu:~/Bureau$
```