Compte Rendu

TP9

Encore des threads + exercice récapitulatif

Exo 2 Tp8:

Principe :

L'objectif est de créer un programme capable de collecter des données de températures à partir de plusieurs capteurs et les stocker dans une matrice. Pour jouer le rôle des capteurs de données, on utilisera des threads qui vont générer un nombre aléatoire compris entre 0 et 32 toutes les secondes. Et toutes les 10 secondes, on cherche la température maximale pour l'afficher dans un dossier "stats_globales.txt". Pour ce faire, on utilisera un dup2. Pour sécuriser la lecture et l'écriture des threads, on utilisera un Lock.

Code :

```
import logging

def collectTemp(index,x):#fonction qui va collecter les températures
    colonne=0
    while(True):#on remplit la matrice
        lock.acquire()
        nbA=random.randint(0,32)
        x[index][colonne%10]=nbA#ici le modulo permet de revenir à la ligne
        lock.release()
        time.sleep(1)#on attend 1 seconde
        colonne=colonne+1
    pass


def creatCapt(pnbCapteur,pmatrice):#fonction qui va créer les différents threads qui joueront le rôle des capte
    for i in range(pnbCapteur):#on crée un thread par capteur demandé
        th = threading.Thread(target=collectTemp, kwargs={'index' : i,'x' : pmatrice})#on définit le thread qui
        th.start()#on lance le thread


#def waitCpt():#fonction qui va permettre d'attendre l'execution de tous les threads
   # main_thread = threading.currentThread()
    #for t in threading.enumerate():
        #if t is main_thread:
            #continue
        #t.join()


def findMax(pmatrice):#fonction qui va trouver le maximum des températures
    tempMax = 0
    tempMax=max(pmatrice)#on récupère le max de la matrice
    print("maximum")
    print (tempMax)#on affiche le max directement dans le fichier grâce au dup2

def genererCollect(nbArg, pmatrice):#fonction qui va lancer la collecte de températures
    for i in range(nbArg):
        lock.acquire()
        matrice2 = numpy.copy(pmatrice[i,:])
        lock.release()
        tr = threading.Thread(target=findMax, kwargs={'pmatrice':matrice2})#on lance le thread qui lance findMa
        tr.start()



lock = threading.Lock()#création du lock pour gérer la protection sur l'action des threads
nbCapteur= 5#nombre de capteur
matrice = numpy.zeros((nbCapteur,10),int)#on crée une matrice de 5*10 remplis de zéros
fich=os.open("stats_globales.txt",os.O_CREAT|os.O_TRUNC|os.O_RDWR)#on ouvre le fichier demandé en lecture et é
os.dup2(fich,1);#on échange la sortie standard avec le fichier, cela permettra d'écrire dans le fichier
creatCapt(nbCapteur,matrice)#on lance la création des capteurs pour relever des températures
while(True):#on entre dans une boucle infinie
    time.sleep(10)#on attend dix secondes
    genererCollect(nbCapteur,matrice)#on commence la collecte
```

Vérification :

```
maximum
31
maximum
30
maximum
29
maximum
29
maximum
32
maximum
27
maximum
29
maximum
32
maximum
30
maximum
30
maximum
32
maximum
32
maximum
30
```

On obtient bien un fichier contenant le maximum des températures relevées toutes les 10 secondes.

Exo 1 Tp9:

Principe :

Le principe est le suivant : on souhaite faire un programme capable de multiplier deux matrices carrées avec des threads qui calcule chaque élément en parallèle. Puis une fois cette matrice obtenue, on la remultiplie avec elle même pour obtenir la matrice au carré. Pour la synchronisation, on utilise un objet de type Barrier, il permet d'attendre qu'un certain nombre de threads est fini leur exécution pour continuer.

Code :

```
import numpy
import sys
import os
import threading as thr



def threadExec(matriceA,matriceB,x,y,matriceMul):#cette fonction va calculer une valeur de la nouvelle matrice
    valeur=0
    for i in range(len(matriceA)):
        valeur=valeur + matriceA[x][i]*matriceB[i][y]#on calcule la valeur d'un élément de la matrice
    matriceMul[x][y]=valeur#on l'ajoute
    bar.wait()#on attend les threads


def creatThread(pNb,matriceA,matriceB,matriceD):#Cette fonction permet de créer les threads pour calculer la m
    for i in range(pNb):
        for j in range(pNb):
            th = thr.Thread(target=threadExec, kwargs={'matriceA' : matriceA, 'matriceB' : matriceB, 'x' : i,'y
            th.start()


matriceA=numpy.array([[1,2],[1,2]])#on initialise la matrice A
matriceB=numpy.array([[4,4],[1,4]])#on intialise la matrice B
matriceFin=numpy.array([[0,0],[0,0]])#on initialise la matrice Fin qui sera le résultat de la multiplication
print(matriceA)#on affiche les deux matrices
print(matriceB)
bar = thr.Barrier(5)#on crée la barrière qui devra attendre 5 threads avant de s'ouvrir
creatThread(2,matriceA,matriceB,matriceFin)#on lance la création des threads
bar.wait()#on attend les 5 threads pour afficher la matrice Fin
print(matriceFin)
bar.reset()#on reset le nombre de thread à attendre sinon la barrière reste ouverte, comme si on a refermé la b
matriceFinCarre=numpy.array([[0,0],[0,0]])#on initialise la derniere matrice Fin au carré
creatThread(2,matriceFin,matriceFin,matriceFinCarre)#on lance le calcul
bar.wait()#on attend la fin des 5 threads
print(matriceFinCarre)#on affiche la matrice finale
```

Vérification :

```
desportes@desportes-VirtualBox:~/Bureau$ python3 Exo1tp9.py
[[1 2]
 [1 2]]
[[4 4]
 [1 4]]
[[ 6 12]
 [ 6 12]]
[[108 216]
 [108 216]]
```

On obtient donc une matrice qui est le résultat de la multiplication des deux puis une autre qui est son carré.