Projet PPC : The Energy Market

I. Introduction

Le but de ce projet est de simuler le fonctionnement d'un *energy market* à l'aide du multiprocessing et du multithreading. Dans cette simulation, il y aura plusieurs foyers qui consumeront de l'énergie et en produiront. Cela permettra au marché de calculer le prix de l'énergie (en fonction de l'offre et de la demande) et de distribuer l'énergie disponible aux différents foyers. De plus, le programme simulera des facteurs externes (comme la météo, des tensions diplomatiques, une pénurie de carburant, etc.) qui feront eux aussi évoluer le prix de l'énergie.

II. Conception

La simulation contient 4 types de processus :

- *home* : chaque maison correspondra à un process. Les maisons peuvent communique via des *message queues* avec le *market* et avec les maisons voisines. L'énergie qu'il souhaite consommer et l'énergie produite seront calculées aléatoirement. Ensuite, si l'énergie souhaitée est supérieure à l'énergie produite, la maison achètera ce qui lui manque au marché ; sinon elle vendra son surplus d'énergie soit au marché, soit aux maisons voisines. Ces valeurs seront envoyées au *market* toutes les 2 secondes.

- *external* : c'est un process child de market. Il simule en continu les éléments externes aléatoires qui viendraient perturber le marché. Pour cela, il envoie des signaux à *market*, selon le pseudo-code suivant :

```
Tant que (vrai):
        Entier n = nb prenant aléatoirement la valeur 0, 1, 2
        Si n==1 alors:
                envoyer à market le signal 1 correspondant à l'événement 1
        Sinon si n==2 alors:
                envoyer à market le signal 2 correspondant à l'événement 2
        Fin Si
Fin Tant que
```

Si n=0, cela signifie qu'il n'y a pas eu d'événements externes, alors rien ne se passe.

- *weather* : ce process a accès à une mémoire partagée créée dans le main (*weatherAttributes* de type Array). Il écrit en continu dans cette mémoire deux paramètres générés aléatoirement : la température et le taux d'humidité.

  Tant que (vrai):
  température = nb généré aléatoirement
  humidité = nb généré aléatoirement
  weatherAttributes[0] = température
  weatherAttributes[1] = humidité
  Fin Tant que

- *market* : il s'agit d'un process multi-threadé. Chaque thread sera connecté à un process de type *home* et communiquera avec ce dernier à l'aide d'un *message queue*. Ce process récupère les informations envoyés par les foyers (l'énergie consommée et l'énergie produite) . Il a aussi accès à une mémoire partagée depuis laquelle il récupère les *weatherAttributes*. De plus, il reçoit les signaux envoyés par *external*. Avec tous ces paramètres, il calcule le prix de l'énergie (à l'aide de la formule donnée dans le sujet). Le prix sera re-calculé toutes les 2 secondes.

On notera, sur le schéma page 3, que les moyens de communication sont en général mis en place par le processus « père » :

- *main()* crée la mémoire partagée que *weather* et *market* utiliseront pour communiquer entre eux. Il la leur passe en paramètre.
- *market* passe son PID en paramètre à *external* pour que ce dernier puisse lui envoyer des signaux.
- *market* crée les Queues pour que ses différents threads puissent communiquer avec le thread principal, et les passe en paramètre à ses threads.
- *market* crée les messageQueues pour que les différents *home* puissent communiquer avec lui.

Les seules exceptions à cela sont les processus *home* qui communiqueront entre eux en sachant que les numéros d'identification des messageQueue pour les *home* vont de 131 à 140.

IV. Début d'implémentation en Python

Ce code implémente les processus *market*, *external*, *weather*. Il reste à implémenter les *home* et les communications qu'elles auront entre elles selon leur Energy Trade Policy.

```
"""EnergyMarket.py

A multi-independant-processes simulation about the energy market."""

from multiprocessing import Array, Process
from signal import signal, SIGUSR1, SIGUSR2
from sysv_ipc import IPC_CREAT, MessageQueue
from os import kill, getpid
from random import random
from time import sleep
from queue import Queue

marketKey = 221
nMarketThreads = 5
externalEvents = [0, 0]

def weather(weatherAttributes):
    print("[%d] Weather : Init" % getpid())
    type(weatherAttributes)
    weatherAttributes[0] = 0
    weatherAttributes[1] = 0
    while True:
        sleep(4)
        weatherAttributes[0] = 20 + 5*random()      # Temperature
        weatherAttributes[1] = 10 + 30*random()     # Rain
        print("[%d] Weather update : T = %.2f and R = %.2f" % (getpid(), weatherAt-
tributes[0], weatherAttributes[1]))

def external(marketPID):
    print("[%d] External : Init" % getpid())
    while True:
        sleep(6)
        if int(2*random()):
            signal(marketPID, SIGUSR1)
            print("[%d] External event : SIGUSR1 sent." % getpid())
        if int(2*random()):
            signal(marketPID, SIGUSR2)
            print("[%d] External event : SIGUSR2 sent." % getpid())

def market_handler(sig, frame):
    global externalEvents
    if sig == SIGUSR1:
        externalEvents[0] = 1
    elif sig == SIGUSR2:
        externalEvents[1] = 1

def market(weatherAttributes):
    print("[%d] Market (main) : Init" % getpid())
    global marketKey, nMarketThreads, externalEvents
```

```
    # launch external process
    print("[%d] Market (main) : launching external process" % getpid())
    externalProcess = Process(target=external, args=(getpid(),))
    externalProcess.start()

    signal(SIGUSR1, market_handler)
    signal(SIGUSR2, market_handler)

    # weatherAttributes is already set and ready to use
    # externalEvents is already set and ready to use

    # launch nMarketThreads queues and threads associated
    queueToSon = [ Queue() for k in range(nMarketThreads)]
    marketThreads = [ Thread(target=marketThread, args=(marketKey+k, queueToSon[k]))
for k in range(nMarketThreads)]
    homesValues = [ 0 for k in range(nMarketThreads) ]

    while True:
        sleep(2)
        homesValues = [ queueToSon(k).get for k in range(nMarketThreads) ]
        print("[%d] On the market we have e1 = %d\te2 = %d\tT = %.2f \tR = %.2f" %
(getpid(), externalEvents[0], externalEvents[1], weatherAttributes[0], weatherAt-
tributes[1]))

def marketThread(marketKey, queueToMother):
    queueToHome = MessageQueue(marketKey, IPC_CREAT)
    try:
        message, t = queueToHome.receive(block=False)
        value = message.decode()
        value = int(value)
    except NotAttachedError as e:
        value = -1
    queueToMother.put(value)
    # after some time
    queueToHome.remove()

if __name__ == '__main__':
    print("[%d] Main process : Init" % getpid())
    # creating shared memory
    weatherAttributes = Array('d', range(2))

    marketProcess = Process(target=market, args=(weatherAttributes,))
    weatherProcess = Process(target=weather, args=(weatherAttributes,))

    marketProcess.start()
    weatherProcess.start()
    print("[%d] Main process : Done" % getpid())
```