PPC Project

Lorenzo CAMARGO
Léa JUSKIEWENSKI

The Energy Market

December 18th, 2018

This report describes the implementation plan of a multi-process and multi-thread project aiming to simulate the energy market. Depending on homes buying, selling or giving energy, on weather and on external random evenements, our programm represent the fluctuations of the energy price.

Processes Overview

1. **Home:** We will have a defined number of Home process. Each home has a house policy which direct its actions : buying, selling or giving energy. A home can as well be in a position of need, asking for a donation or willing to buy energy on the market.
2. **Market:** The market is the process computing the price energy, depending on the informations provided by all the other processes.
3. **External:** The external process simulates random events happening during a random time duration which impact the energy price.
4. **Weather:** The weather process gives the temperature value, as well impacting the energy price.

IPC Implementation

- Home <-> Home

They will communicate via message queue. They are used to distribute operations or variables through multiples processes. In Python we import the library “Queue” to initialize a queue, and from “multiprocessing” we’ll use the Event method to signals that the queue was modified:

```
queue = Queue()
Data_ready = multiprocessing.event()
```

To use them in each Process, we wait for a signal (blocking method), and then we can read from the queue:

```
Data_ready.wait()
value = queue.get()
```

In these queues we will put a message with the amount of Energy they will give to another Home that is in need.

- Home <-> Market

We will define 2 messages queues: one for the communication from Home to Market, and another for the other way. In the first one we will put messages from Homes indicating what they want to do with their Energy situation (sell, give, buy, wait for a giver and the amount of Energy). In the second one, the Market will put messages corresponding to the answer of a demand of a Home (say if they were able to sell, buy or if there is a home able to give...and the amount of Energy).

- External <-> Market

The External process uses Signals to indicate that a random event happened to the Market process. This method can be used because the External process is a child process of Maret. To send a signal, in the External process we’ll import the “os” library and use the method:

```
os.kill(os.getppid() , #number of the signal)
```

From the parent, to handle a signal we need to create a function that handles signals, from the library “signal”:

```
Def handler(sig, frame):
        If sig == signal.#type of signal send
                #do something
```

The parent process will always be listening thanks to a “signal” method that we’ll put into the Market:

```
signal.signal(signal.#type of signal, handler)
```

The Market and the External have the same list of events, each event is assigned to as different signal number (30,10,16, 31,12,17 SIGUSR signals).

- Weather <-> Market

This process need to send the value of the current temperature via a Shared Data. We'll import from the "multiprocessing" library to use "value" method. Each process can interact with the Value object to read or write in it: Weather will write the temperature in the Value and Market will read from it. Will create it by:

```
Temperature = Value('d' , #value of temperature)
```

And use with:

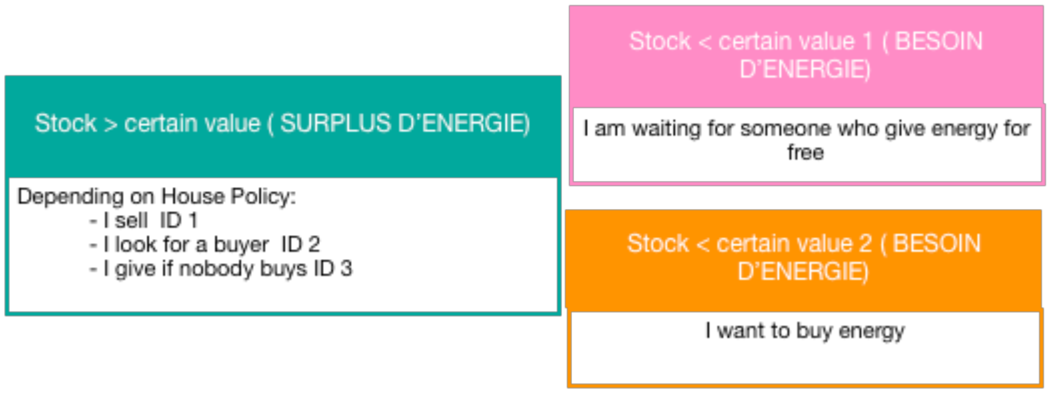
```
Temperature.value = #change
#somthing(Temperatur.value)
```

Energy Consumption/Production

Each home defines a random consumption rate between [x ; y], re-valuated at each home energy calculation, a fixed production rate, and a stock value, which represent the amount of energy available.

This stock value is read from a message queue, where the market store the value left for each home after a transaction (cf. next part). Reading in a message queue is a blocking operation, which mean that the the process will wait for having a message in the queue. This is coherent for us because it is nonsense if the process compute a new demand while the last one has not been handle yet.

Then, the home process compute the value of stock (energy remaining after consumption and production). And depending on this value, add a message to a queue.



Stock > certain value (SURPLUS D'ENERGIE)

Depending on House Policy:
- I sell ID 1
- I look for a buyer ID 2
- I give if nobody buys ID 3

Stock < certain value 1 (BESOIN D'ENERGIE)

I am waiting for someone who give energy for free

Stock < certain value 2 (BESOIN D'ENERGIE)

I want to buy energy

The message as sent as a table like this one :

ID maison	ID policy	Montant d'énergie à donner/vendre

Handling the Energy Market - Market Threads

- Transaction threads

The Market process will create 5 threads, each one will handle an Energy transaction. Each thread reads from the Message Queue of messages from Home, to access the queue they need a to acquire a Mutex Lock to prevent threads from reading at the same time. The message is interpreted : if a hom want to sell Energy the thread will send to the other queue a message, an array of int, in this form:

ID House	Amount of Energy (negative, positive or zero)

If a Home wanted to sell/give Energy this thread will send negative amount of Energy sold (the same amount that came in the request message). If a Home wanted to buy/get Energy we'll send a positive amount. If a Home want to search for a taker, the thread will store the message and keep looking for a "I want Energy" message from other Home for a certain period of time. I this time runs out, it will just send a message containing a negative amount of Energy.

The difference between selling and giving Energy only comes up when we want to calculate the Energy Price: when buy or sell we influence it, when we give or take it doesn’t affect the price. In that case the transaction threads need to communicate with the Energy Price Thread to inform when a transaction takes place and the amount of Energy Sold/Bought. The communication between these two kind of thread take place via a Global variable.

- Energy Price Thread

This thread will actualise the Energy Price in a determined frequency. As our Market process is always listening to signals from External, we need to store each External event in an array. These events are not infinite, they have a time to live, measured in number of periods. The External process signals he name of the event, a coefficient to quantify the vent and a Time to Live. Each period, an Energy Price process’ method verifies that each event of the array storing the events should be alive or not, and destroys an event if so. Each period he will read from the Shared Memory shared with Weather to take the Temperature, we read from the array storing the events and we take the Amount of Energy Bought variable and from the Amount of Energy Sold variable. We then inject all these values in the price calculator function we’ll define to redefine the market price.