Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Lab work – n°1
Processus and monitor synchronization

Documentation

The concept of "monitor" can be used to synchronize processes using conditions (in the **multiprocessing** module). See the documentation available in Moodle.

Exercise 1 - Producer/Consumer Model - Version without synchronization

- Retrieve the code named *m1_prodCons_base.py* provided in Moodle.
- Run the application for different parameter values.

Examples of execution:

```
python3 m1_prodCons_base.py 3 3 1
python3 m1_prodCons_base.py 2 2 2
python3 m1_prodCons_base.py 4 4 4
```

and see the problems with unshared variables.

- Modify this program to replace the constants **`nb_times_prod`** and **`nb_times_cons`** (number of deposits made by a producer and number of withdrawals made by a consumer) with two additional parameters given at runtime.
- Modify this program to use shared variables between processes. Note the access conflicts on the shared variables.

Exercise 2 – Producer/Consumer Model – Basic version

From the provided code, **add the monitor synchronization** - using mutual exclusion locks and conditions - needed to implement the basic producer-consumer version (seen in class and TD) in which withdrawals occur in the order of deposits. As in the tennis court example, build a monitor in a python class.

- Run the application for different configurations to ensure the validity of the results obtained.

[Code to upload on Moodle]

Exercice 3 – Producer/Consumer Model - Alternate Deposits

Modify your code (**keeping** the previous version) to implement the version where the producers alternately drop their messages in the buffer (see V2 of the TD). For the tests, use an even number of producers.

[Code to upload on Moodle]

Exercise 4 – Producer/Consumer Model - On Demand Withdrawals

Modify your code (**keeping** the previous version) to implement the version where consumers request to withdraw a message of a certain type and their withdrawals must be done in the order of the deposits (see V3 of the TD). We will no longer force the alternation of deposits.

Exercise 5 – To continue... Producer/Consumer Model - Dual Deposits

Modify your code (**keeping** the previous version) to implement a version where at **each** deposit, a producer deposits two messages in an (**unbreakable** and) **consecutive** way in the buffer (so, the deposer() function will have to be adapted). The removal of messages will be done in the same way as in the basic version, i.e. in the order of the deposits (final exam June 2017).

Reminder: If the displays are too fast, it is possible to delay the execution of a process for a few microseconds or nanoseconds using the primitives:

time.sleep(secondes)

See the online manual for their use:

https://docs.python.org/fr/3/library/time.html#time.sleep

We can use a randomly generated value (see the functions random.rand and random.seed) to vary the waiting times from one process to another.

https://docs.python.org/3/library/random.html

But, **be careful**, the timeout is not there to solve the problems of concurrent access to shared variables. In other words: any execution of a parallel application must give a consistent result without timeout!