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

Lab work – n°2
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.

Exercice 1 – Reader-Writers

We consider parallel activities (processes) that simulate readers and writers having read or write access to a common file. Reads can be done in parallel but writes can only be done in mutual exclusion.

The behaviors of the processes are as follows:

A Reader	A Writer

Loop on {
…
`start_read()`
Read the shared file;
`end_read()`
…
}
	Loop on {
	…
	`start_write()`
	Modify the shared file;
	`end_write()`
	…
	}

Ensuring monitor synchronization, write the **`start_*`** and **`end_*`** operations so that a writer gives priority to another writer and has priority over pending read requests (V2 of the TD).

Test these operations by writing an application in which L readers and R writers coexist (L and R can be parameters of the application).

Note: A code skeleton is provided in the file *tp2_lectred_base.py*

[Code to upload on moodle]

Exercice 2 – Management of a small road

Parallel activities (processes) are considered that simulate the behavior of vehicles traveling in a certain direction. We also consider a portion of a single lane on which, in order to avoid collisions, only vehicles going in the same direction can circulate.

The behavior of the vehicles is as follows:

```
Vehicle(direction)
Loop nTimes on {
        Drive normally on the large road in the direction direction
        enter_road(direction)
        Drive on the small road in the direction direction
        exit_road()
}
```

Write the **Road** monitor to synchronize the accesses to the single lane so as to allow an unlimited number of vehicles to travel in the single lane as long as they go in the same direction

Test these operations by writing an application in which N1 vehicles going in one direction and N2 vehicles going in the other direction coexist (N1 and N2 can be parameters of the application).

Examples of possible tests:

N1 = 2, N2 = 5, nTimes = 2

N1 = 0, N2 = 3, nTimes = 2

N1 = 4, N2 = 0, nTimes = 3

Note: A code skeleton is provided in the file *tp2_vu_base.c*

[Code to upload on moodle]

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!