Clock based Mutexes, exercise 2

# Overview

The goal of this exercise is to fully understand the Lamport and Agrawala algorithms, as well as to implement them in a distributed system (simulated on one machine with different processes).

The application we built for this is a simple app where “Heavy processes” (Process A and Process B) each spawn some “Light weight processes”, which will access a shared resource. That shared resource simply prints on a console the messages it receives.

Here is the architecture diagram on which we based our application (as given in the slides).

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

As you can see on the diagram, the processes will directly interact with the shared resource. The shared resource does not handle concurrency in the different process trying to access it. Therefore, the light weight processes have to synchronize themselves in order to have the desired execution.

Note : given that we didn’t use a token ring system for the exercise n°0, this exercise also let us learn how to implement those.

# Execution

Here is a screenshot of the display process when the execution was done :

Une image contenant capture d’écran, motif, monochrome, tissu

Description générée automatiquement

And here is the bottom of a heavy process display, once the execution has been done :  
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

# Heavy processes

The heavy processes are very similar to one another. Here is the main method in HeavyProcessA.java:  
Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

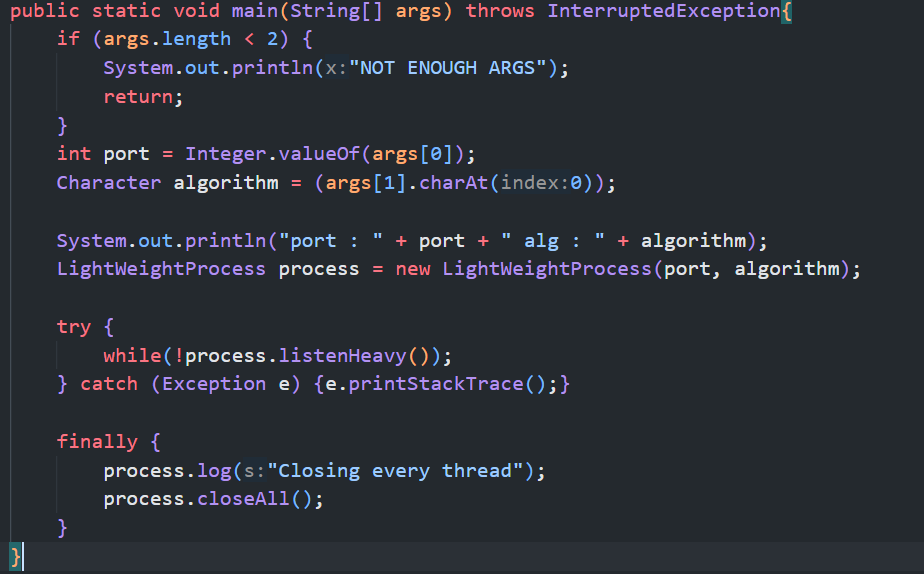
When the heavy processes are created, they will make a connexion between them. Then, they will each spawn subprocesses, and orchestrate the connexions among their respective subprocesses.

Here is the code snippet that ensures the different connexions are made among light weight processes :  
Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Once all the connexions are made properly, the heavy process with the token tells its subprocesses to write to the shared resource (the display console). Once it (heavy process) knows that they are done with the shared resource, it sends the token to the other heavy process, which does the same.

# Light Weight Processes

The class LightWeightProcess is used by both heavy processes, and depending on the information it receives, it will either become a Lamport node or an Agrawala node. Here is the main method in the LightWeightProcess class :  


# Difficulties

During the development, the biggest difficulty was to debug the subprocesses, since using the C debugger was a painful process. We decided to debug them by making them print on the HeavyProcess console.

Some synchronization issues also occurred in the management of the threads within a LightWeightProcess, and we don’t fully understand how we repaired those issues (but we did).

# Conclusion

In this exercise, we implemented 2 different mutual exclusion algorithms : the Lamport and Agrawala. The execution of this project starts 3 different shells : Process A, Process B and the main screen.

The Processes A and B each spawn 3 sub Processes that will use one of the algorithms to synchronize the access to the Main screen. Processes A and B communicate by using a token ring to know who’s sub processes can access the Main screen.