Task details :

- **Current Implementation version status :**

  - Implemented Bully Algorithm.

  - 4 different processes start separately and execute the algorithm behavior perfectly.

  - Processes communicate with each other using RMI in java.

  - Only 1 jar file, 1 class that implements 1 interface.

  - No threading used in the process simulation.

- **How to run it :**
  - You can use the exported jar file (double clicking it would be enough).

  - You can build the project and run the Process class, inside eclipse (and it will generate the same output).

  - Process can only be terminated by using (ctrl + c) inside the terminal, closing the terminal would produce unexpected behavior (the process would be terminated without unbinding it from the registry).

  - Process can be restarted by opening a cmd in the jar directory and write the following command (without quotes) :
  `"java -classpath bin -Djava.security.policy=java.security.AllPermission -Djava.rmi.server.codebase=file:classDir/ MainPackage.Process 2 1919"`

  Where 2 is the process ID , it can be 1,2,3 or 4 , according to which process to run, and 1919 is the port number for the registry to run on, if changed, all processes must be restarted with the new port number.

  - Some output statements (each process talking to leader, leader reply, ..) commented so that you can see better output for important statements (election starting, winning, leader down, ..) , you can uncomment these lines inside the code for full view, you will find these lines with a comment “Uncomment the following line” before each of these lines.

- **Assumptions :**
  - Assuming that using a Shutdown Hook is not considered as using Threading in the communication simulation process.

  - Assuming that the cmd is enough for displaying the output.

- **What can be done to complete the task :**

- Give the GUI framework a look, implement a simple GUI.

- A window containing a controller, with binding this controller to the output variable, or raising events with listeners for displaying output, for each process.

- **Method used for communication :**

After a deep comparison between sockets and RPC (Remote Procedure Calls), I preferred using RPC, as each process can act as both server and client, a process can send a message to a certain process, or broadcast a message to all running processes.
All processes register themselves with a unique ID in a registry, all processes can get this registry and get objects by ID from it.
Having the registry running on a certain port, it can be accessed by all processes.
Each process can then send an object of it through a different port, to be registered in the registry.
Now all processes can communicate by calling any method inside other processes through the registry.
Terminating a process is accompanied by an unregister command, so that the terminated process no longer exists in the registry.
All this implemented in java using RMI (Remote Method Invocation) for the remote procedure calls.
While in sockets, We need to handle which sockets started and used, specify protocol, format messages .. and so on.
But for RMI, it handles all the networking details, all I had to watch for where the ports, It can be considered as a higher level layer over the sockets.