Distributed mutual exclusion

R1

We have implemented peer nodes by giving each node a server part and a client part.

- The server part is responsible for listening for messages from other clients and executing code depending on the type of message received.
- The client part is responsible for sending requests to the server depending on the current state of the client. The client can either start an election, say it is the new coordinator, request the token and release the token.

When a coordinator is found, any node can at any time request for the token, which is used to enter and use the critical section.
The critical section is emulated by sleeping for some time to show a process taking some time and logging a message "Using the critical section" to show that something is happening.

R2

We implemented the critical section system with a token which the server and nodes keep track of. The server has a “Coordinator token” which tracks whether a client can be granted the token. The nodes themselves have a token boolean which tracks if they have the token or not. The server holds a queue of access requests from nodes, and gives access one at the time. When access to the critical section is granted, it can only grant further tokens once it receives a release from the node. Currently the server will wait indefinitely, until it receives this response, this means that if the node crashes, the token is lost and a new election has to be made, so the token is generated again, which is a problem for the system as whole.

R3

The system is generally lively and fair, whenever a node requests access to the critical section, the request is put into a queue inside the server. The order of the queue is purely determined by first come first served principle and requests are kept until it’s processed. Thus it follows liveliness, as other requests won’t overtake

previous requests, but if the server node crashes or somehow gets reelected, the requests in queue will currently be deleted and forgotten, which is a flaw in the way we have implemented the system.

The algorithms

We have implemented two algorithms, one for elections and one for distributed mutual exclusion.

- Elections: To implement elections we have used the bully algorithm, this one works by a node starting an election. This is done by sending an election message to all bigger nodes. Whenever a node does not get a response back from all bigger nodes, it can assume it is the biggest node and send a coordinator message to all other nodes.
  Now every node will now send requests to the coordinator node for access to the critical section.
- Distributed mutual exclusion: To implement distributed mutual exclusion we have used the central server algorithm where the current coordinator is granted the role as the server node. Every node will now send requests to the server node which keeps a queue of requests and grants access to every node in a fair way. When a node has been granted access it can now enter the critical section and will afterwards release the access when it is not needed anymore.

Git repository

https://github.com/ITU-DISYS23-GROUPPandemicHalfPals/Hand-in4

Appendix

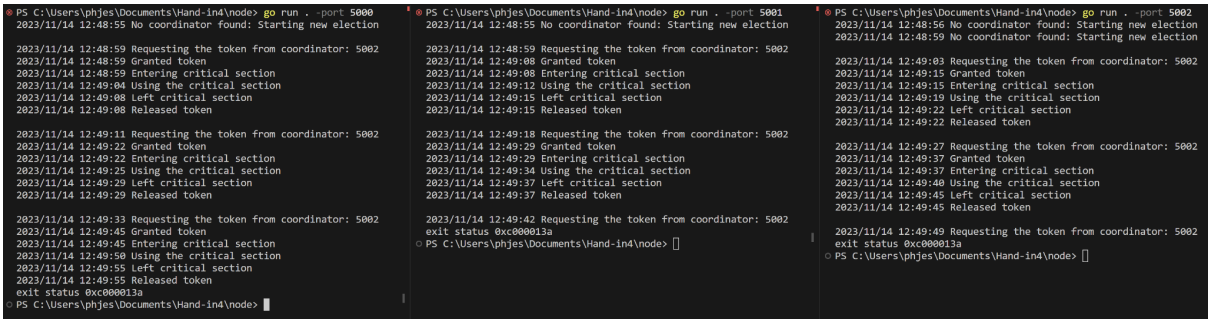

Figure 1: Log from three nodes at once. It shows that three independent node instances can run and can only access the critical section only one at once, as seen from the log dates.

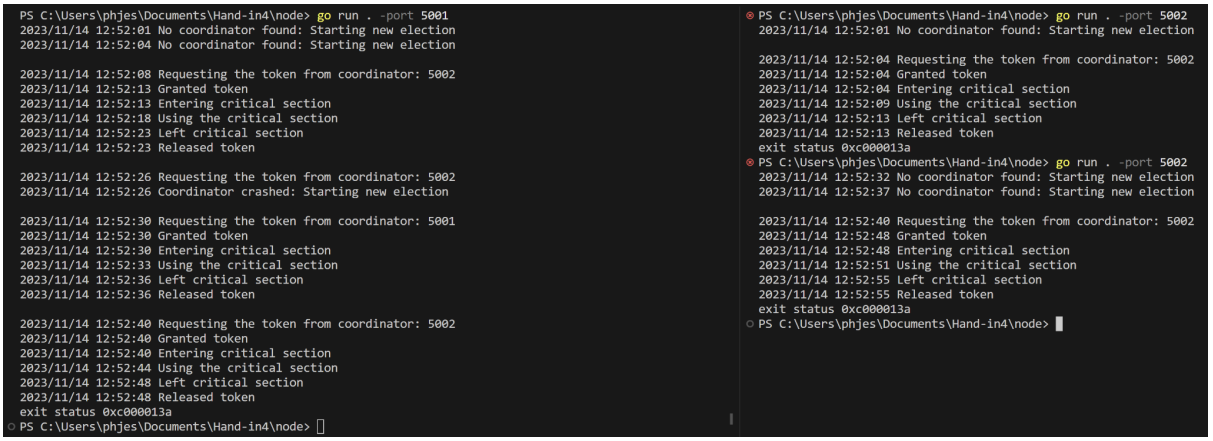

Figure 2: Log of two instances of nodes. It can be seen that if the coordinator crashes or somehow stops or a node joins or rejoins, the collective system will reelect a new coordinator.