FOURTH PRACTICAL

RICART-AGRAWALA MUTUAL EXCLUSION ALGORITHM W/

FRANCISCONE LUIZ DE ALMEIDA – M304083
EDUARDO FÚNEZ FERNÁNDEZ – M303494
DAVID RUIZ PELÁEZ – M303488

Introduction

The objective of this assignment is to implement a distributed algorithm for mutual exclusion. To achieve this task, we use the Ricart-Agrawala algorithm in addition to JGroups as communication resource.

This algorithm is based in the communication via messages between different nodes. There will be two kinds of nodes:

- **Requesting node:** this node will request to enter the critical section
- **Receiving node:** this will be every other node which is receiving the request from the requesting nodes

As we know, the coexistence of more than one process in the critical section can lead to inconsistencies and data loss.

How does it work?

In this algorithm two type of messages are used: REQUEST and REPLY. The communication channels are assumed to follow a FIFO order.

A node sends a REQUEST message to all other sites to get permission to enter the critical section. Nodes will send REPLY messages to give its permission to enter the critical section.

A timestamp is used to determine the priority of critical section requests: Lower timestamps get higher priorities than higher timestamps. The execution of the critical request is always in the order of their timestamps. These timestamps will be given by a logical clock.

The algorithm will follow the next steps:

1. **Entering critical section (CS):**
    - When a node wants to enter the critical section it sends a timestamped REQUEST message to all other nodes.
    - When a node receives a REQUEST message from a node, it sends back a REPLY message if the receiving node is neither requesting nor executing the critical section. Otherwise, the REQUEST message will be denied by said node.
2. **Executing the critical section (CS):**
   A node will only enter the critical section if it has received the REPLY message from all other sites.
3. **Releasing the critical section (CS):**
   Upon exiting the critical section, the node sends REPLY messages to all denied REQUEST messages. This informs nodes that the critical section has been released.

Ricart-Agrawala algorithm requires $2 * (N - 1)$ messages per critical section execution being N the number of nodes (A REQUEST and REPLY message from every node minus the messages from the node that enters, executes and releases the critical section).

Implementation

To develop this assigment we used the language Java along with some third-party libraries such as JGroups. The project contains tree main classes: Peer, PeerData and PeerGUI.

- **Peer**: Classe used to process the requests and replies of the nodes. This class contains the core of Agrawala's algorithm and extends the ReceiveAdapater from JGroups. To achieve right results the class must override the receive method of the superclass, in which we'll process each possible message received by the node. There are 3 possible states that a node can be, which are: ready (when a node is ready to receive requests to CS), waiting (when a node made request for the other to enter the CS and is waitings for their replies in order to enter the CS) and in CS (which the node is executing the critical section and node another node can do that).

- **PeerData**: It contains the info of each node to be sent as a message allowing the nodes to communicate and to reply properly.

- **PeerGUI**: A class that provides a basic user interface for interaction and demonstration of the problem. The figure below (figure 1) shows the example of the GUI. The "current state" and "last replyed to" fields are self-explanatory, otherwise, the "Ack Queue" and the "Req Queue" fields represent the Acknowledgment Queue to count how many nodes have replied to the request which controls the access to the Critical Section. On the other hand, the Request Queue represents the queue of nodes waiting for the current node response since it couldn't reply at the right moment.

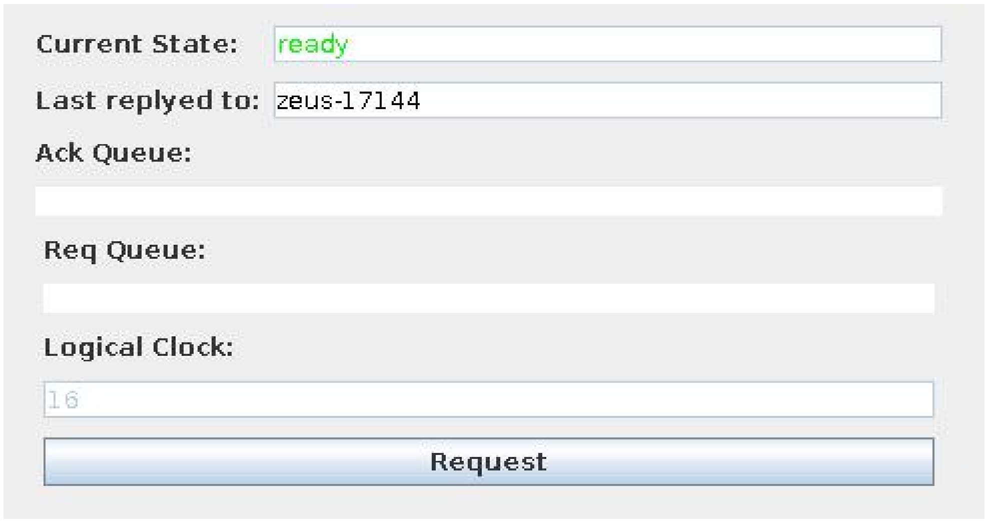
Current State: ready
Last replyed to: zeus-17144
Ack Queue:
Req Queue:
Logical Clock:
16

Figure 1: Graphical User Interface of the System

Ricart-Agrawala Algorithm

This section aims to explain in detail the execution of Ricart-Agrawala's algorithm. To start the process, a node from the network (**figure 2**) desiring to enter the CS asks all the peers for permission to do it. To do so, the peer sends a request message to all the other nodes in the network (**figure 3**) and if they are available or have lower priority than the current node, a reply message is sent allowing the execution. Each response is added to an Acknowledgment Queue (AckQueue) and once all the nodes have replyed the queue size is match with the size of the network and the peer may execute the Critical Section (**figure 4**).

If another node makes the request while a node is "waiting" or "in the critical section" then the request is put into a Request Queue, and only when the waiting/CS is finished the response is sent to the nodes. Once the node finishes the critical section it releases the execution and starts the replying process if there is any request in the queue.

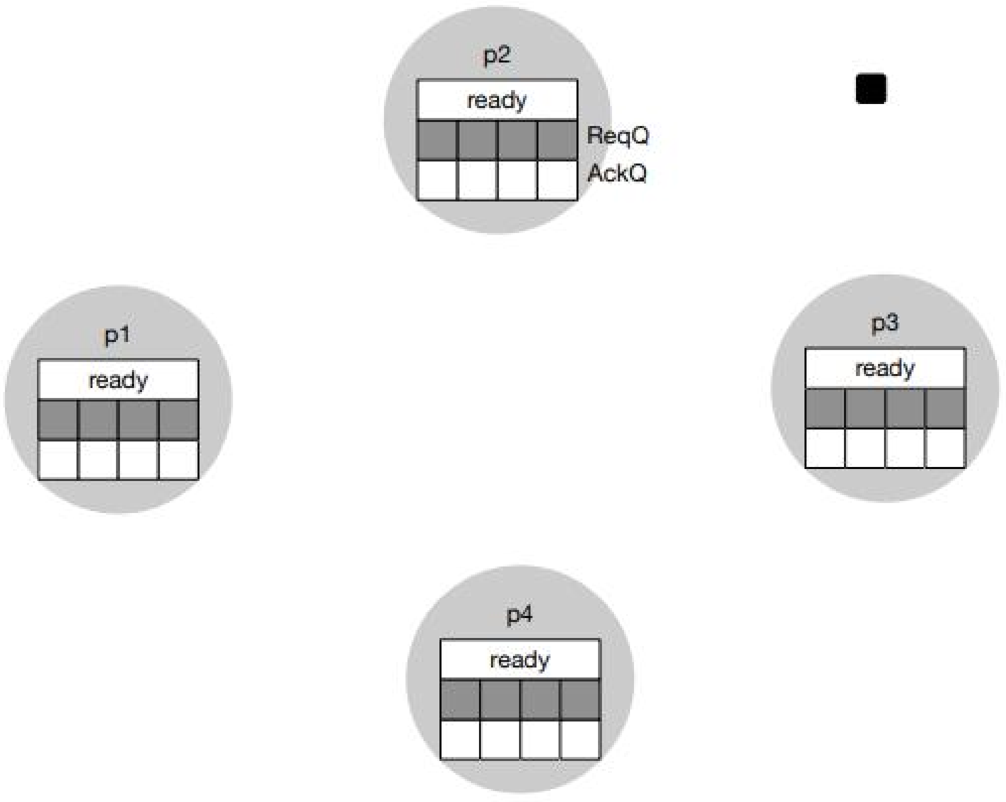

Figure 2: Peers connecting to the network.

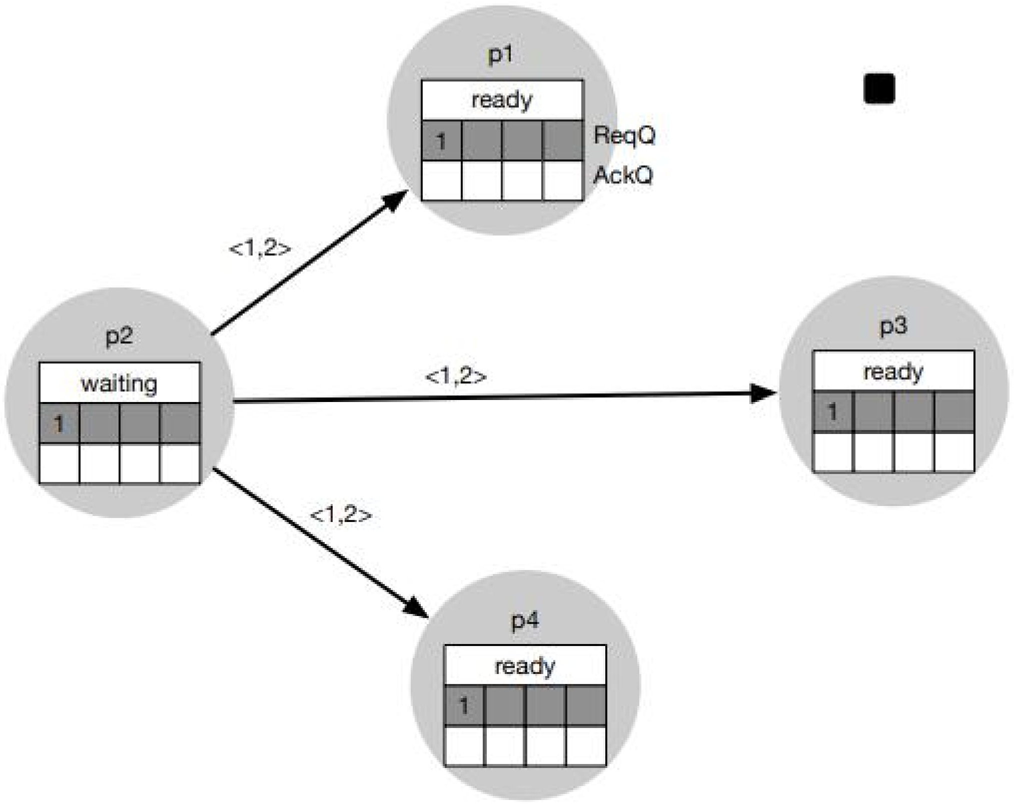

Figure 3: Peer requests CS access to other nodes.

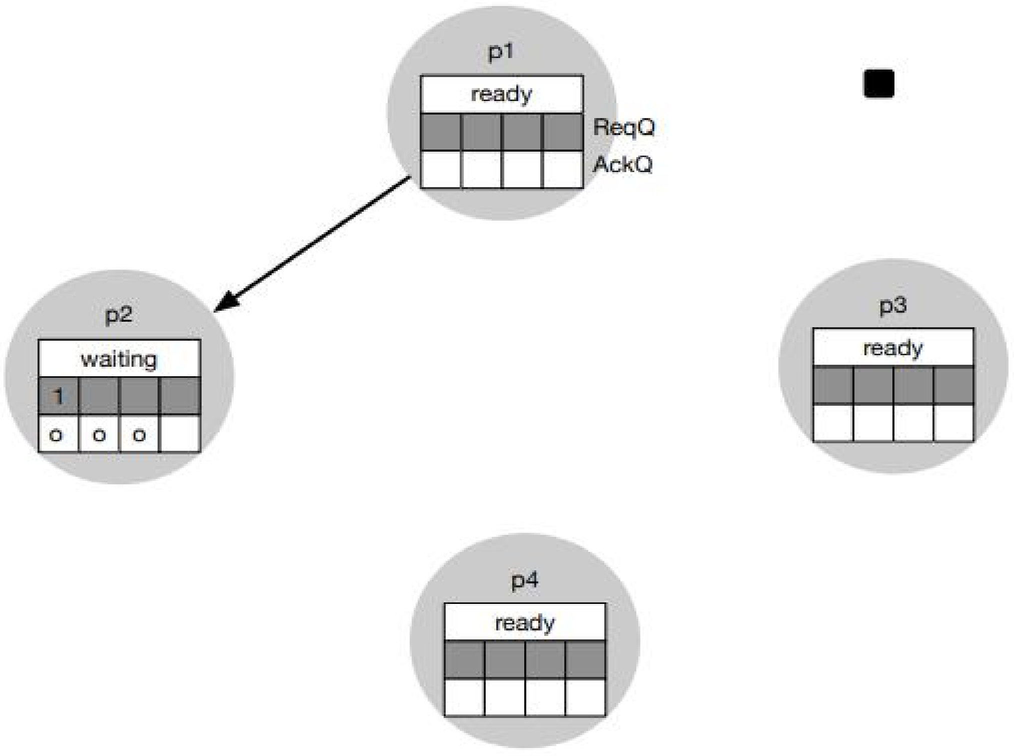

Figure 4: Each node replies granting CS access.