Eric Pelletier (260895863)

Zachary Siciliani (260845463)

COMP 512

Programming Assignment 2: Total Order Using Paxos

**Paxos Implementation**

Starting with the broadcastTOMsg which is called by the process trying to send a message to all of the processes while maintaining a total order. The process starts by broadcasting a “propose” message to all of the processes. This message consists of an Object array where the first item in the array is a string “propose” and the second item is a ballot Id. The ballot Id corresponds to the process Id plus a sequence number which is the smallest multiple of 10 greater than the max ballot Id that the process has seen so far. Therefore, if the greatest ballot Id seen so far is equal to 13 and process 1 wants to send a proposal message, it will send out a ballot Id of 21. This ensures that two processes will not be sending out a proposal message with the same ballot Id. After sending out the proposed messages, the process waits until a majority of processes have either accepted or denied the proposal through a promise message that will be described later on. If the majority of processes have denied the proposal, the process will retry sending a proposal message with the ballot Id incremented by 10. The process retries sending the proposal until it finally gets accepted which helps ensure that the process’ move gets sent to the processes. If the majority of processes have accepted the proposal, the process will first check if processes have returned an already accepted value in their promise message. If so, the process will use the accepted value of the last message received with an accepted value in the upcoming “accept” message. This helps cover a failure case where the proposer has crashed after sending an “accept” message and was not able to send out the “confirm” message to tell the process to deliver the message to the application layer. The process also keeps in mind that the value that it wanted to send got overwritten and will restart at the proposed step once the accepted value retrieved is installed by all processes. Following the majority of processes having accepted the proposal for leader, the process will send out an “accept” message with the value for the processes to accept. It will then wait until the majority of the processes have accepted or denied the value sent. If the majority of processes have denied the value, then the process will restart the entire procedure starting at propose with a higher ballot Id. If, on the other hand, the majority of processes have accepted the value, then the leader will broadcast a “confirm” message with the value to send to the application layer to all the processes.

For the acceptTOMsg which is called by the application layer to figure out what is the next message in the total order. This method starts by checking if there are any messages that are sent from a proposal process. The process does this with the use of a message buffer system that will be described later. The messages that are of concern for the process in this method are the “propose”, “accept” and “confirm” messages. If the message that is being analyzed corresponds to a “propose” message, the process checks the ballot Id in the message and will return a “promise” message containing any currently accepted values to the sender promising that it will consider this process as the leader only if the ballot Id is greater than the maximum ballot Id seen so far. Or else, the process will send back the current highest ballot Id seen so far indicating that it will not consider this process as the current leader. If the message corresponds to an “accept” message, the process will send back an acknowledgement message and will store the value sent in the message (to send in any propose messages that might arrive before being sent to the application layer) only if the ballot Id in the message is still equal to the maximum ballot Id seen so far. If not, the process will deny the “accept” message by sending back a denial message. If the message corresponds to a “confirm” message, the process will then send the player number along with the value in the confirm message to its application layer by returning from the acceptTOMsg method.

To maintain total order, the leader process will attach a sequence number in its “accept” message which is sent to processes to “accept” the proposed value. The sequence number gets incremented by 1 when a process sends a message to its application layer. When the process receives the “accept” message, it checks if it is bigger than the highest sequence number that it has seen so far and returns an acknowledgement if this is the case and the other criteria are respected. If the sequence number is not larger than the value seen so far, the process returns a denial message along with the highest sequence number seen so far.

As mentioned earlier, our paxos implementation uses a message buffer system since the processes are not able to identify by themselves which messages they are receiving with the given implementation. Therefore, this message buffer system gets called by the processes whenever they are expecting a message. The message buffer system first checks if there are any messages that have been read from the gcl layer but that have not yet been read by the process itself. The message buffer system does this by keeping count of the total number of messages of the given type and the messages read from that type. The type of messages are “propose”, “promise”, “accept”, “acknowledge” and “confirm”. The broadcastTOMsg is only concerned with the promise and acknowledge messages and the acceptTOMsg is only concerned with the propose, accept and confirm messages. If a process wants a certain type of message and that message read counter is equal to the total number of messages in that message type’s buffer, then the system will read from the gcl until it has read a message from the requested type. For example, if the process is in the broadcastTOMsg function and is waiting for the “promise” messages from the other processes, it will call the message buffer system to get these messages. This way, the process is not handed any messages of the wrong type. In addition, the acceptTOMsg functions a bit differently in which it will request any messages of types “propose”, “accept” and “confirm” at any given time. This allows the process to answer to a “propose” message if it is waiting for a “confirm” message from another process. Also, it allows for the process to continue execution with a different process in the case of a leader crashing before sending its “accept” or “confirm” messages.

To ensure that all the processes have a fair chance of pushing their move, at the end of the broadcastTOMsg, the process waits for 500ms. This allows the chance for other processes to gain control of the leadership. In addition, after every message is sent, the process waits 100ms, allowing the other threads in the process a chance to execute code instead of being blocked trying to read a message that was supposed to be sent by the same process.

**Contribution**

Eric came up with the architecture and design to handle the paxos implementation and failure cases while Zachary came up with the design to handle the total ordering.

Eric implemented the paxos architecture and design, and Zachary helped debug it.

Eric wrote in the report about the architecture and design choices for the paxos implementation while Zachary wrote about the design choice to handle the total ordering.