1. Introduction

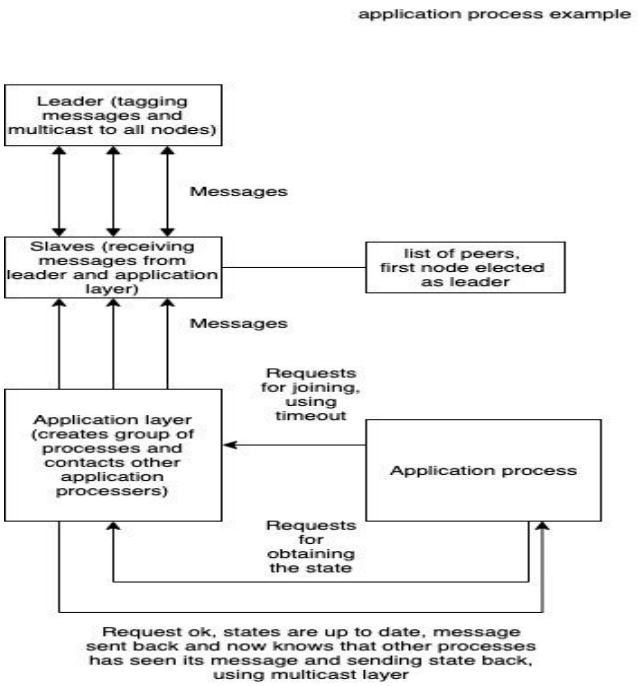
The topic in this assignment regards group membership service that provides atomic multicast. A group membership allows for processes to join the group and send messages to it with regards to reliability and ordering of the group. This is called multicast communication. Atomic multicasting is achieved when the message is delivered to every correct member, or no member at all. The fundamental idea of group communication is that a process only executes one multicast operation in order to send a message to each of a group of processes, as opposed to executing multiple send operations to individual processes.

All nodes that wish to multicast a message will send the message to the leader, and the leader will commit a basic multicast to all members of the group of processes. In case of a leader failure (e.g. by losing network connection), a new leader is elected, namely the first node (e.g. a slave) in a queue of nodes (e.g. queue of slaves).

A new node that wishes to enter the group will contact any node in the group and request to join the group. The leader will determine when the node is to be included and deliver a new view to the group.

2. Main problems encountered

Grasping the whole idea of group membership communication in the code was quite challenging. As per usual, a graph was drawn to facilitate the understanding of the concept (see below).



The following command only spawned one worker at one point:

- test:more(5, gms3, 1000).

Solution: The workers failed to join the group due to a bug in the code. By changing {join, Wrk, Group, Peer} to {join, Wrk, Peer} and by incrementing N once we broadcast a message as well as calling the leader function again, it worked. Figuring out when and where to increment N was a bit hard, but if one thinks about “receive” in the leader function, it is quite obvious that once a new incoming message is asked to be multicasted, we need to increment the sequence number (N). Otherwise, we are casting and creating message duplicates once we call the leader function again, and making it impossible to decide whether to discard messages or not in the I < N clause of the slave function.

3. Evaluation / Results

The result of gms1 is different workers changing color in the GUI once a new message is multicasted. However, when a leader fails, a new leader is not elected and no more messages can be multicasted to the other nodes.

In the results of gms2, a new leader is elected, but this leader is not exposed by the last leader’s final message. Therefore, the new leader is not synchronized with the other slaves, creating failures in the message ordering.

In gms3, this is handled by passing the last message sent by each leader to the first slave in the slave queue. Once a leader crashes, a new elected leader can now multicast the previous failed leader’s message to all nodes and thus maintain the message ordering.

4. Conclusion

The assignment gave an overview in how group communication with atomic multicast works. It was especially hard to interpret the code compared to the theory behind the mentioned communication method. Group communication with atomic multicast allows for communication among processes with only one message sending operation executed, rather than multiple message sending operations, creating redundancy.