Harsh Vardhan Vashistha
UF10897

Project 2 Report

My implementation has class **Clock** implementing Logical clock and class **Communicator** implementing **multicast** as well as **unicast communication**. **Communicator** also implements **Berkley Algorithm** and **Total Ordering**.

Mutual Exclusion is implemented as Token ring algorithm and takes support from Communicator class for the group information.

Communicator class first discovers the multicast group membership, initiated from master and also record unicast address for all process and distributing the same among each process. Based on this group membership information, a process with smallest ID is than chosen as new Leader.

(Algorithm Implementations, as in my program)

Berkley Algorithm :
Implementation: Communicator::berkleySync()

1. Leader sends self clock (multicast).
2. Each process calculates the difference and send that back to leader (unicast to leader).
3. Leader on receiving differences, calculates the average.
4. Leader then calculates the necessary offset for each process to synchronize and send this offset to the respective process (unicast to process).
5. Each process add offset to its clock.

Total Order (Lamport algorithm), (All send and receive are multicasts)
Implementation : Communicator::setOrderedRecv()

1. Each process marks its outgoing message with timestamp <Logical Clock>.<Process id> This outgoing message is immediately added into local queue <Priority min queue on timestamp>
2. Receive buffered is queried for incoming message and the message is added to the local queue prioritized on timestamp.
3. A process only acknowledges a message if the message has smallest timestamp in the queue, and it is not already acknowledged by the process.
4. If the message with smallest timestamp has received acknowledgement from all the processes, it is pushed into an ordered queue. Other threads can read from in total ordered sequence.
5. Repeat till no more messages are being received.

Above algorithm is written in the most simplistic form, which does not include packet drops in multicast. In Communicator class, above algorithm unwinds into a much larger case by case scenario accounting for packet drops and re-request for the messages.

Following two scenarios took almost all of my time to solve and are not related to project, but I did implement few other OS concepts because of them.

1. Setting a **timeout on multicast sockets** makes them stop receiving messages. Couldn't find a documented case available on this. I ended up using pselect() to implement timeout after wasting too much time because of this problem.
2. **Packets were getting lost in multicast communication** and in considerable amount. This is most likely because of UDP protocol's unreliability and wifi networks. Lamport's algorithm in very dependent on reliable communication and hence, this was a problem. I had to implement techniques to handle the packet losses and make the algorithm robust to such losses.

Mutual Exclusion (Token Ring Algorithm):
Implementation: One of the case in process.cpp:main()
Communicator collects multicast group information during initiation, this includes ID and unicast address for each process.
Using process IDs, I implemented token ring.

1. If process has token goto step 2, else wait to receive token
2. Read file, increment the counter in the file, write back and close
3. Pass the token to the process having immediately next higher ID. (Last process pass to first).

Execution instructions: See **readme.md**

My own execution output is present in the folder "**Output**" for 3 simultaneous process.

These outputs are also evidences for:
Berkley Sync, total ordered receive and normal receives (No total ordering).
Files:

"./Output/Berkley and Total Order (Includes non ordered)/berkley_total_order_*.txt"

Berkley Sync and Mutual Exclusion.
Files:

"./Output/Berkley and Mutual Exclusion/berkley_mutual_exclusion_*.txt"