# “PROJECT -2 REPORT”

# 5306 - DISTRIBUTED SYSTEMS

**Spring 2019**

Submitted by :

Harshil Patel – **1001717222**

Sreenidhi Timmaiahgari – **1001721679**

# Assignment 1:

**Lamport's Algorithm**

To implement Lamport's logical clocks, each process Pi maintains a local counter G. These counters are updated as follows.

Steps:

1. Before executing an event Pi executes Ci = Ci + 1.
2. When process Pi sends a message m to Pj , it sets ms timestamp ts (m) equal to Ci after having executed the previous step.
3. Upon the receipt of a message m, process Pj adjusts its own local counter as Cj=maxCj, ts (m), after which it then executes the first step and delivers the message to the application.

**Totally Ordered Multicast**

Totally Ordered Multicast is implemented as follows:

1. Apply Lamports algorithm
2. Every message is timestamped, and the local counter is adjusted according to every message
3. Each update triggers a multicast to all servers
4. Each server multicasts an acknowledgement for every received update request
5. Pass the message to the application only when:
    a. The message is at the head of the queue.
    b. All acknowledgements of this message have been received
6. The above steps guarantees that the messages are in the same order at every server, assuming message transmission is reliable the algorithm ensures that the events occurred at each process in the same order in all processes. The delivery restrictions enable the ordering of the messages.

Totally ordered multi-cast takes a lot of network bandwidth as messages as well as acknowledge is multi-casted to other processes.

**IMPLEMENTATION:**

**Requesting process:**

1. Pushing its request in its own queue (ordered by time stamps)
2. Sending a request to every node.
3. Waiting for replies from all other nodes.
4. If own request is at the head of its queue and all replies have been received, enter critical section.
5. Upon exiting the critical section, remove its request from the queue and send a release message to every process.

**Other processes:**

1. After receiving a request, pushing the request in its own request queue (ordered by time stamps) and reply with a time stamp.
2. After receiving release message, remove the corresponding request from its own request queue.
3. If own request is at the head of its queue and all replies have been received, enter critical section.

**WHAT DOES THE PROGRAM DO?**

- Client0 acts as server to both client1 and client 2 Client1 is the server to client2 and client for Client0 Client2 is clients for both.
- The acknowledgements from the clients are named as ack0, ack1 and ack2. Once the multicasting begins the timestamps are noted by each process and the time stamp matches.

**ISSUES ENCOUNTERED:**

1. The timer was required to be maintained and updated. This was only possible by maintaining a thread separate from the main running threads. Even with the all the extra overhead, the timer was well adjusted as expected.
2. Since, the actual implementation takes very little, it was important to introduce a slight delay in sending out the events, so that the network is not overly used.

## Assignment 2:

**Vector Clock:**

Vector Clock allows us to implement easier totally ordered synchronization between devices.

The property of Vector Clock VC[j] can be described as below.

1. V Ci [j] is the number of events that have occured so far at Pi.
2. If V Ci [j] = k then Pi knows that k events have occurred at Pj .

Based on above-mentioned properties, Vector Clock behaves as follows.

1. Before sending a message, Pi executes, V Ci [i] = V Ci [i] + 1 (1)
2. When process Pi sends a message m to Pj , it set m’s (vector) timestamp ts (m) to V Ci after having executed the previous step.
3. When node Pj receives a message from node Pi with ts(m), it delays delivery until:
    a. ts(m)[i] = V Ci [i] + 1ts(m)[k] <= V Cj [k]foranyk <> i
4. Upon the receipt of a message m, process Pj adjusts its own vector by setting
    a. VCj [k] = maxV Cj [k], ts(m)[k]
5. for each k and delivers the message to the application. This step will allow us to simply obtain the feature of totally order multicast.

**IMPLEMENTATION:**

1. The assignment was implemented using python. The server and client are separate processes to be launched from the terminal.
2. Every process maintains its latest timestamp and acts accordingly when new events are created or received from others.
3. The real client sends a new event to the dummy server, it then calls any process at random, and creates a new event. The process then handles the rest of the process.
4. The events are executed in the same order as their timestamps.
5. To verify this, the process output their events on their own individual files. The files at the end of the testing, proved to have maintained the same order of output.

**ISSUES ENCOUNTERED**

1. The major issue was maintaining and timestamp of every process. This was resolved by maintaining an array of timestamp and encoding the timestamp with the event message when broadcasting to other events.
2. The other issue was verify the solution was correctly implemented. This was resolved by outputting the execution sequence to individual files. The sequence was also logged to the terminal, so the sequence could be correctly deduced.

## ASSIGNMENT 3

**Token Ring Algorithm**

Token ring algorithm is one of the mutual exclusion algorithms.

1. In this algorithm the processors are arranged using a logical ring and each processor knows its neighboring processor.
2. They pass the token along the ring based distributed system and only the process that has the token has the right to use the shared resource.
3. If the process that has the token wishes to enter the shared resource, it used the token to open the shared file and when it finishes it will pass the token to the next process.
4. If the process doesn't want to enter the shared resource, it just passes the token to the next process.

Token ring Algorithm is simple to deploy. However, it does not work if tokens are lost. Furthermore, it is difficult to detect whether the system has lost the token or not.

**IMPLEMENTATION:**

1. This was implemented by using a custom Lock mechanism created for the project. The mechanism maintains and blocks the calling process until the lock is held by any of the other process.
2. This could be verified since, the output to the shared files shows that each process increments its own count one after the other i.e every process receives the lock after one operation.

**ISSUES ENCOUNTERED:**

1. Using the lock system provided by the language, was not very effective. Hence, a custom solution was required to block the calling process, and keep it on hold until the holding one has performed its operation.
2. The solution truly justifies the token ring behavior, since the caller is blocked until the previous process has released the lock.

## KEY LEARNING

We learnt to implement different synchronization methods to achieve total order multicasting between devices and the importance of locking scheme.

## REFERENCES

- Lecture notes of Dr. Jia Rao.
- Distributed Systems Principles and Paradigms Edition 2