Distributed Systems, Project2

CSE 5306, Fall 2021

Contributors:

PAVAN KUMAR CHAPARLA – 1001865675, pxc5675@mavs.uta.edu

POLA MANOHAR – 1001980283, mxp0284@mavs.uta.edu

05th November, 2021.

REQUIREMENTS:

- Assignment-1(35pts) Implement totally ordered multicasting using Lamport's algorithm. Each process conducts local operations and numbers them as PID.EVENT_ID. After each operation is done, a process multicasts the event to all other processes in the distributed system. The expected outcome of this assignment is that events occurred at different processes will appear in the same order at each individual process. To realize such a total order of events, each process maintains a buffer for received events and follow the rules on slide 19 in Lecture 6 when delivering the events. In this assignment, the delivery of events is simply printing them on screen, in the format of CURRENT_PID: PID.EVENT_ID. HINT: You may use two threads in each process to handle the communication and deal with message delivery, respectively. Refer to the multithreaded server example discussed in class. The communication thread enqueues updates in a buffer, from where the delivery thread enforces a total order of events. The communication thread is also responsible for sending acknowledgements for received messages.

- Assignment-2(35pts) Implement the vector clock algorithm to enable causally-ordered events in the distributed system. Similar to the previous assignment, we use multiple processes to emulate multiple nodes. Assume that all nodes are initiated to the same vector clock. After completing a local operation, each process sends its updated vector clock to all other processes. Follow the steps on slide 34 in Lecture 6 to implement the vector clock algorithm.

- Assignment-3 (30pts) Implement a locking scheme to protect a share file in your distributed system. While we use processes on a single machine to emulate the distributed system and there are shared-memory synchronization primitives available, you are required to implement one of the following distributed locking schemes: centralized, decentralized, distributed locking or the token ring algorithm. When a process acquires the lock, it simply opens the file, increments a counter in the file, and closes the file. Assume that all processes keep requesting the lock until successfully acquiring the lock for a predefined number of times.

CONCEPTS COVERED IN ASSIGNMENT '1':

- We have analysed the project and worked together to start with the usage of Lamport Algorithm and Total Ordering between processes.

-We went through certain course materials online to get the in-depth knowledge of the concept of Lamport-Algorithm before starting with the implementation and shared the tasks among us.

-Created three Client class files which act as client for itself and server for the other two files will send and receive data in packets.

-This data once performed with the operation will be notified to other processes i.e other Clients using the Lamport concept of notifying.

CONCEPTS COVERED IN ASSIGNMENT '2':

-In this part we have used the same example used in Assignment 1 and implemented the logical clock operation between the processes while communication.

```
            long timestamp = System.currentTimeMillis();

            System.out.println(String.format("%s :
%s",receivedDataArray[i],dateFormat.format(logicalTimeStamps.get(i))));
```

CHALLENGES DISCUSSED WHILE DOING ASSIGNMENT 1 AND 2:

-Using the different concepts like hashmaps to maintain the order of the events was typical

-The logical Clock with the processes seemed a bit easier when discussed but took us some time while implementing.

CONCEPTS COVERED IN ASSIGNMENT '3':

-We have implemented the locking scheme for the file locktest.txt on which the fileHandler class gives lock access to certain clientHandlers.

-The Client Runner thread can be called multiply to create the different client classes and each can be used to ask for the access to the file lock.

ISSUES ENCOUNTERED WHILE IMPLEMENTING:

-Initially we have thought of implementing it with building multiple clients with a server. But as the FileLocking class we wrote was capable of looking for the locks we decided to remove the server which was an extra class.

-The multiple clients created can be used to ask for the file lock access which will be dealt by the lock co-ordinator in the FileLocking.class.

-Had some issues while calling from the ClientRunner initially but with combined efforts we could overcome them.