**CMSC 621**

**Advanced Operating Systems**

**Ajinkya Wakhale**

**AN75014**

**Project2 Report-**

**Part 1- Implement Berkeley Algorithm**

Answer: Program name is **sender\_test.c.** For all the parts of the project, I have used one program only. In this, I am using sockets and concept of multicasting group. You can create multiple instances of the program and run it. Each sender\_test.c can be a process each with two threads. Each program running is a process.

1. To run the program, you need to give two parameters in command line arguments. First is rank of the process and second is the message to be multicast which will be used in the part 2. Rank part is necessary only for the first part.
2. Program has two sockets- one is for point to point to communication and other one is for multicasting. Multicasting socket sender structure is mapped to a unique address and unique port which tells that they belong to that multicast group. So, if someone sends a message from multicast socket, it will be received to all the processes in the multicast group.

To allow all the processes to join multicast group, I have given a timer so that in that time, all processes who wants to join multicast group will join. There is the concept of reusability which allows it to use the same port again necessary for multicast.

Steps for first parts are as follows:

1. First, the structure is, all the processes will wait for rank 0 to connect. Once, rank 0 joins the group, all the processes will get a logical clock value based on the random function. After that, rank 0 will send its clock value to all the processes and at the same time will send its port number to all the processes to perform point to point communication.
2. Second, all the non-zero rank processes will receive master clock value and will calculate drift. They will also receive master port number to which they will map their point to point socket. All the processes will send their drift and will send it back to the master.
3. Only master will receive drift values from the remaining processes. Master will calculate the average and will again broadcast the average. So, all processes will receive the average value.
4. All the processes will receive average sent by master. And, they will subtract the average with their difference and finally they will subtract the result to their old clock value to get new clock time. All process will do the same, and finally clock will be synchronized.

**For part 2- To Implement Total Ordering**

In the command line, we are sending, we are sending messages to be multicasted.

There are two threads sender and receiver with their handler. Sender will send the message and receiver will receive the message. Once sender sends a message, it will be multicasted to the group. I am sending the messages in for loop and assigning the value of for incrementor to make the message unique.

To implement total ordering of the messages received, I have used a sequencer program called sequencer.c. Message will be multicasted and then will go to sequencer too. Sequencer port is hardcoded 3210 in code. To make point to point communication with sequencer, all processes will map their point to point socket with the sequencer port.

The sequencer program is just adding a global sequence number to the message and it multicasts the message again. Message will be <M, GS>

In receiver thread, all messages will be received. Order of the message receiving at each process can be different but final message order which must be sent to an application will be according to the sequencer. Whatever order, sequencer receives, all messages should print the message per that order.

In Receiver thread, first it will check whether the message is from sequencer or from any process of Multicast group. If it is from Multicast group, it will add it to buffer.

If the sequencer message comes, it will check whether the message received is already in the buffer. If it is, it will check following condition:

1. Whether, message is there in the buffer
2. Its local clock value+1 is equal to the global sequence number.
3. Whether, message has already been sent or not.

If all the conditions is true, it will send the message to application.

There might be a case, if sequencer message comes before its multicast message, in that case, it will add the message and its sequencer value to its sequence buffer.

And for every multicast message, it will check whether the message is in sequence buffer and the message global sequence value is equal to its local+1. Also, whether the message has not been sent.

If all the condition matches, it will print the message.

There might be a case when all the processes receive messages in different order, sequencer algorithm will make sure they are in total order.

**PS: To store messages in buffer for part 2, I have used array of structures. Its size is 10000. If number of messages are greater than 10000, it will give error.**

**Part 3- Implement distributed locking-**

At the start of the program, I have implemented locking mechanism. There could be multiple processes which are trying to access a file and update it. Only, that process which has lock on that file, will be able to write into that file. If you try to run multiple processes at the same time, values will always be consistent depending upon who has lock and who is trying to change it. There is a file called **counter.txt** which has counter value. If a process runs a program it checks whether the resource is locked or not. If not locked, it will update the counter and will read back. Read back value and updated value will be same.

By this project, I learned a lot of things like Multicasting a message, synchronizing several processes, inter process communication. We learned how to implement several algorithms like Berkeley algorithm for clock synchronization, sequencer algorithm for total ordering. Also, the concept of reusability of sockets. Also, I learned how to implement locking in distributed systems.

Difficulties which I faced while implementing this project was testing. When number of processes are small, messages always comes in order. So, several sleeps have to incorporated in the program to make sure you get different ordering. Also, it was difficult to deal with multicast sockets.