CMSC 621 : Advanced Operating Systems
Project-2

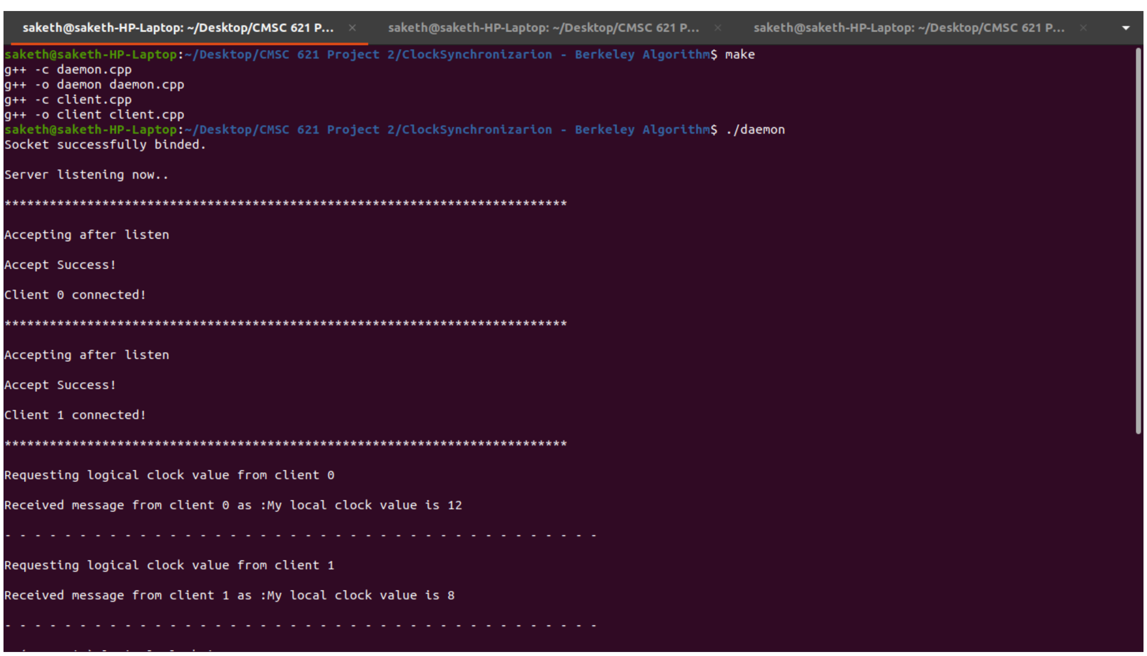
Vangeepuram Saketh Ram
XV45698

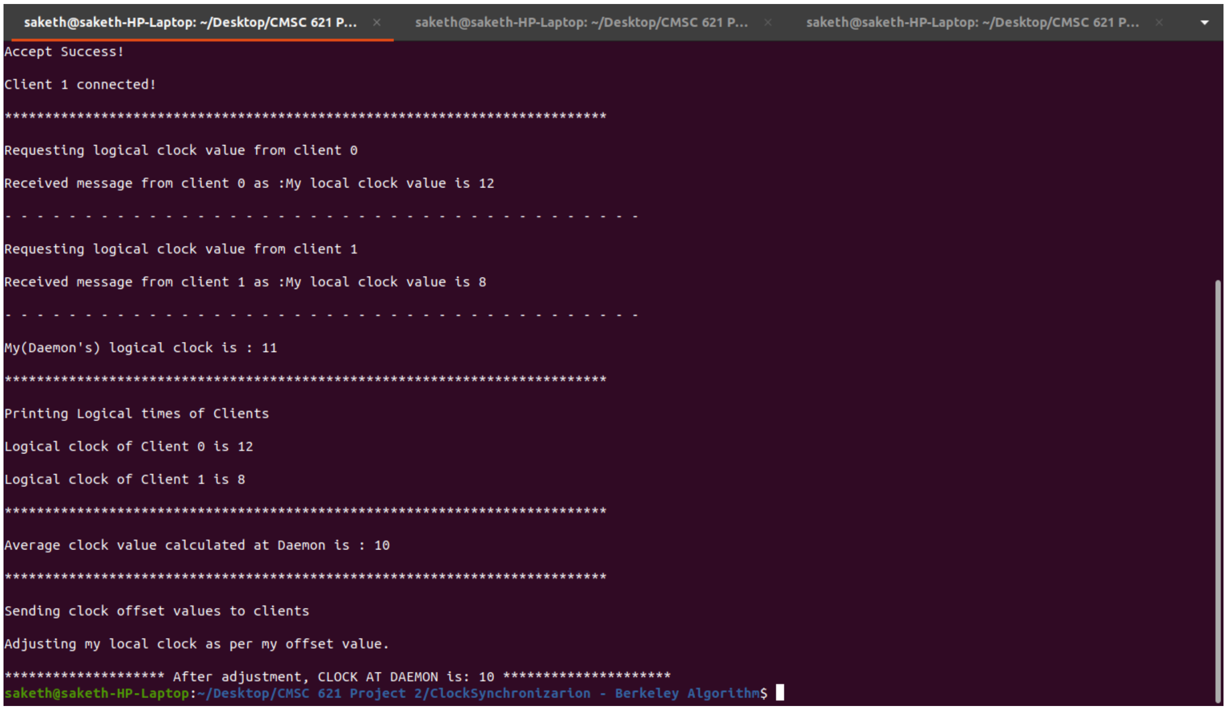
1. Implementing Berkeley's synchronization algorithm

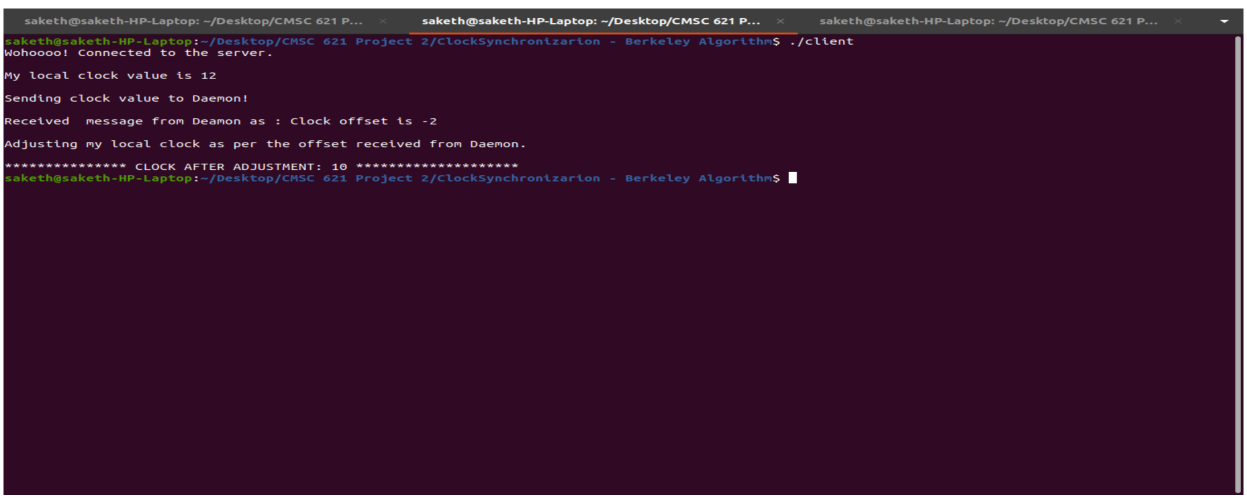
Implemented Berkeley algorithm using C++ to achieve clock synchronization in a distributed system. The algorithm follows the below steps :

- A Master/Daemon node is chosen among all the nodes in a distributed system and remaining nodes act as slaves.
- At first, Daemon sends a message to all the other nodes in the network asking for their respective clock values.
- All the nodes send their clock to Daemon as requested.
- Daemon then calculated the average clock values by including the clock value of itself.
- Daemon then adds the calculated time difference to its local clock and then broadcasts the respective time differences to all the other nodes in the network.
- All nodes then adjust their clocks based on the time difference sent by the daemon.

Please find the screenshots below :
Screenshot 1, 2 shows o/p from Executing daemon.cpp. Screenshot 3 represents executing client.cpp





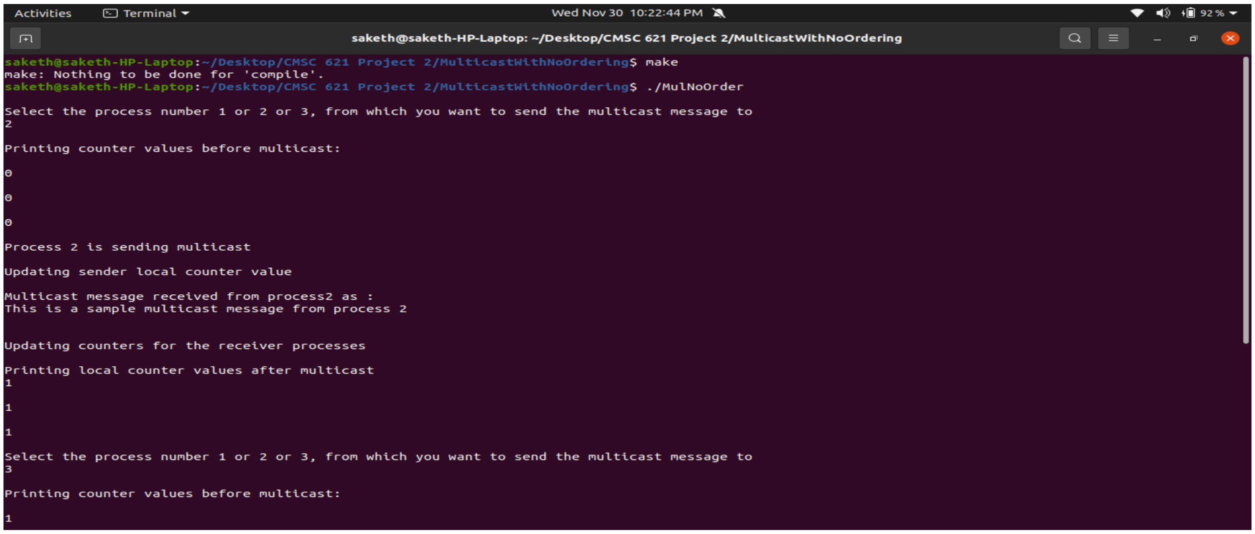


2. Multicast with no ordering

- Implemented multicast programming with 3 clients in a distributed environment.
- All three process would be connected to each other and each process will have two threads -
  - One, for sending the multicast message to all the nodes
  - Other, for listening on the communication channel for any incoming messages.

- Whenever a process sends a multicast message to the group, sender thread will send the message to other two nodes by updating its counter value, to track number of messages/events occurred at that particular process.
- Receiver thread will listen on the communication port and update the local counter values.

Refer screenshot below showing the o/p:



3. Multicast with causal ordering

- Here, all the processes would be connected with each other but we will maintain a buffer queue and a vector for ordering of the messages.
- Each process maintains a vector which has the clock values for all the nodes.
- In the multicast message, the sender sends its own vector to all the other processes. Before sending the vector, it increments its vector index by 1 in the vector i.e, if Pj is sending a multicast then Pj will update as :$Pj[j] = Pj[j] + 1$
- If Pi receives a multicast from Pj with vector as $M[1…N]$ (= $Pj[1…N]$) in message, it will buffer it till the below conditions are met:
  - This message is the next one Pi is expecting from Pj, i.e., $M[j] = Pi[j] + 1$.
  - All multicasts, anywhere in the group, which happened-before M have been received at Pi, i.e., for all $k \neq j$: $M[k] \leq Pi[k]$ i.e., Receiver satisfies causality.

What have I learned?

- Learned why clock synchronization is important in Distributed Systems and how it is implemented using Berkley's Algorithm.
- Learned how to approach and debug a distributed system problem and possible things to keep in mind while designing distributed systems.
- Had a good understanding of different types of communication techniques used in Distributed Systems like FIFO, Causal and total ordering.
- Understood why ordering is required and what is the significance of each ordering technique.

Issues encountered:

- In the Berkeley algorithm, I used character array to receive clock values and then I am converting them into integer values to calculate average and sending back the clock difference(offset) so that other nodes can read the clock offset to adjust their respective clocks.
    - The above steps failed initially as I am not converting the character array to string while reading values.

- The coding part for the multicast causal ordering communication technique was quite challenging for me.