

# CMSC 621 : Advanced Operating Systems

## Project-2

Vangeepuram Saketh Ram  
XV45698

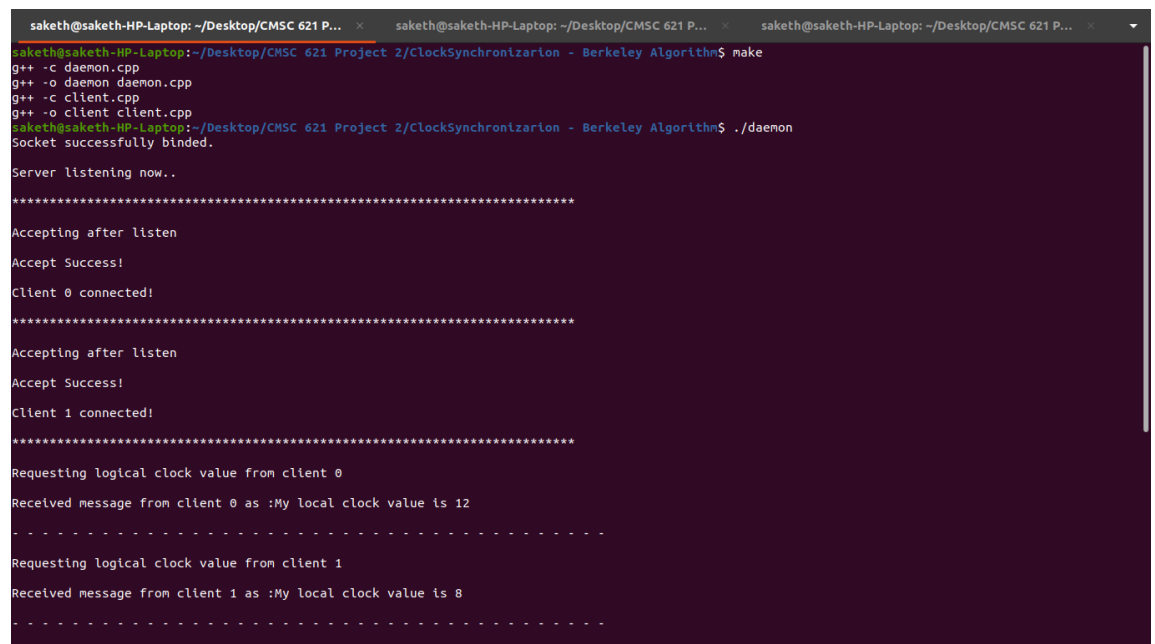
### 1. Implementing Berkeley's synchronization algorithm

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



```
saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/ClockSynchronizarion - Berkeley Algorithm$ make
g++ -c daemon.cpp
g++ -o daemon daemon.cpp
g++ -c client.cpp
g++ -o client client.cpp
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/ClockSynchronizarion - Berkeley Algorithm$ ./daemon
Socket successfully binded.

Server listening now..
*****
Accepting after listen
Accept Success!
Client 0 connected!
*****
Accepting after listen
Accept Success!
Client 1 connected!
*****
Requesting logical clock value from client 0
Received message from client 0 as :My local clock value is 12
- - - - -
Requesting logical clock value from client 1
Received message from client 1 as :My local clock value is 8
- - - - -
```

```
saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x
Accept Success!
Client 1 connected!
*****
Requesting logical clock value from client 0
Received message from client 0 as :My local clock value is 12
- - - - -
Requesting logical clock value from client 1
Received message from client 1 as :My local clock value is 8
- - - - -
My(Daemon's) logical clock is : 11
*****
Printing Logical times of Clients
Logical clock of Client 0 is 12
Logical clock of Client 1 is 8
*****
Average clock value calculated at Daemon is : 10
*****
Sending clock offset values to clients
Adjusting my local clock as per my offset value.
***** After adjustment, CLOCK AT DAEMON is: 10 *****
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/ClockSynchronizarion - Berkeley Algorithm$

saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x saketh@saketh-HP-Laptop: ~/Desktop/CMSC 621 P... x
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/ClockSynchronizarion - Berkeley Algorithm$ ./client
Wohoooo! Connected to the server.
My local clock value is 12
Sending clock value to Daemon!
Received message from Deamon as : Clock offset is -2
Adjusting my local clock as per the offset received from Daemon.
***** CLOCK AFTER ADJUSTMENT: 10 *****
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/ClockSynchronizarion - Berkeley Algorithm$
```

## 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:

```

saket@saketh-HP-Laptop: ~/Desktop/CMSC 621 Project 2/MulticastWithNoOrdering
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/MulticastWithNoOrdering$ make
make: Nothing to be done for 'compile'.
saketh@saketh-HP-Laptop:~/Desktop/CMSC 621 Project 2/MulticastWithNoOrdering$ ./MulNoOrder
Select the process number 1 or 2 or 3, from which you want to send the multicast message to
2
Printing counter values before multicast:
0
0
0
Process 2 is sending multicast
Updating sender local counter value
Multicast message received from process2 as :
This is a sample multicast message from process 2
Updating counters for the receiver processes
Printing local counter values after multicast
1
1
1
Select the process number 1 or 2 or 3, from which you want to send the multicast message to
3
Printing counter values before multicast:
1

```

### 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  $P_j$  is sending a multicast then  $P_j$  will update as  $P_j[j] = P_j[j] + 1$
- If  $P_i$  receives a multicast from  $P_j$  with vector as  $M[1...N]$  ( $= P_j[1...N]$ ) in message, it will buffer it till the below conditions are met:
  - This message is the next one  $P_i$  is expecting from  $P_j$ , i.e.,  $M[j] = P_i[j] + 1$ .
  - All multicasts, anywhere in the group, which happened-before  $M$  have been received at  $P_i$ , i.e., for all  $k \neq j$ :  $M[k] \leq P_i[k]$  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.