ASSIGNMENT NO. 4

Problem Statement :

Implement Berkeley algorithm for clock synchronization.

Tools/Environment :

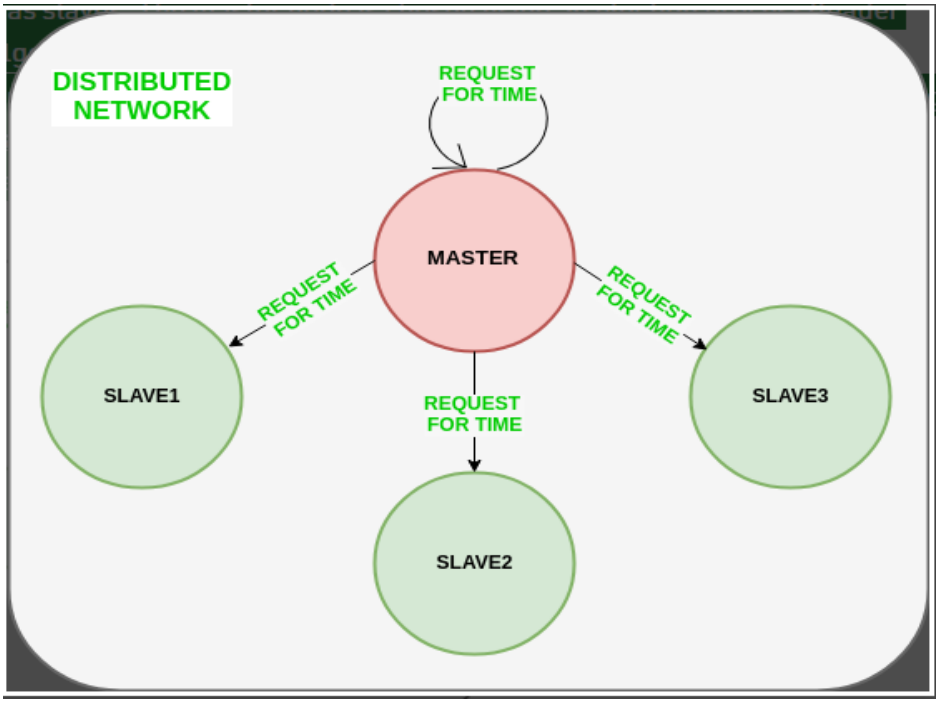
Ubuntu , Turbo C++

Theory :

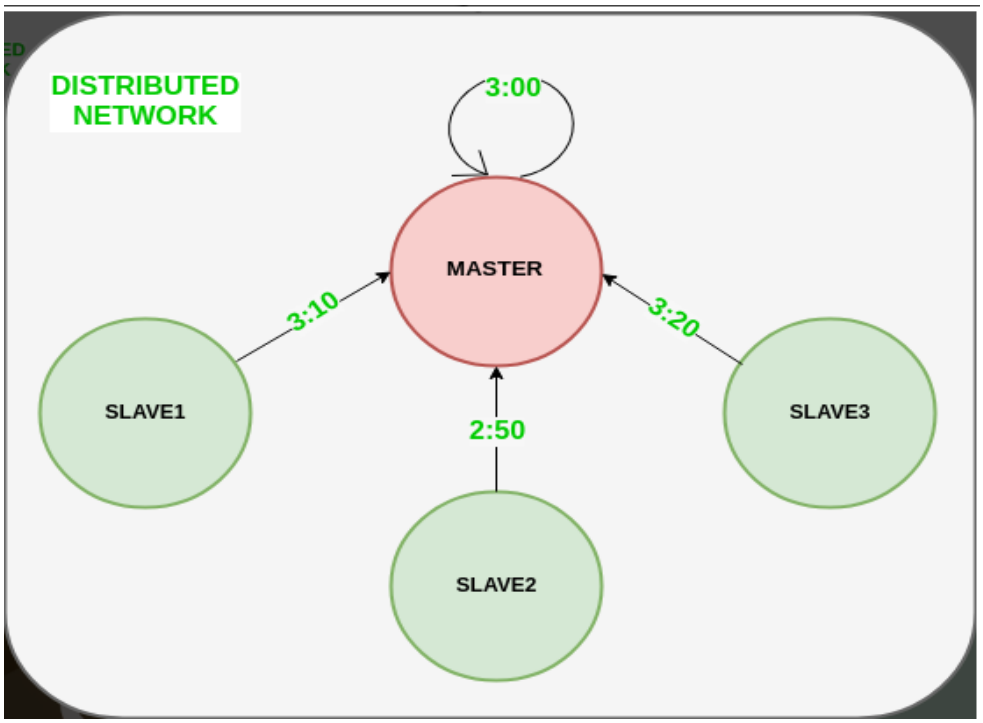
Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

Algorithm

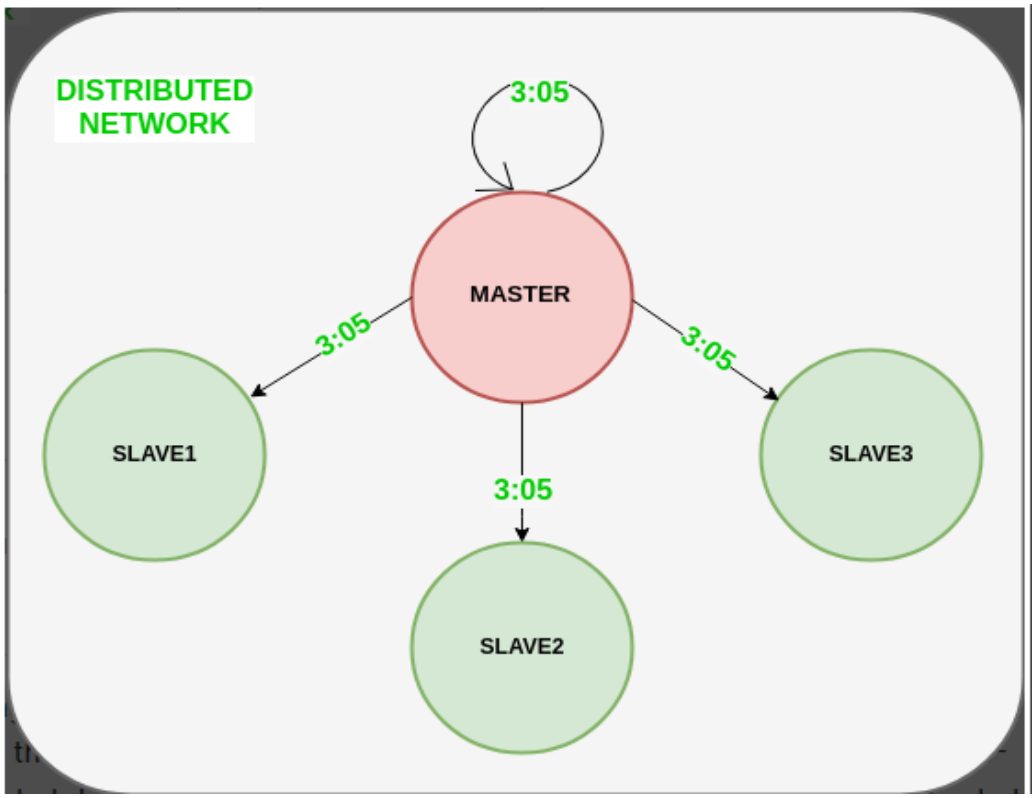
1) An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.
2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.
The diagram below illustrates how the master sends requests to slave nodes.



The diagram below illustrates how slave nodes send back time given by their system clock.



3) Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.



Features of Berkeley's Algorithm:

- **Centralized time coordinator:** Berkeley's Algorithm uses a centralized time coordinator, which is responsible for maintaining the global time and distributing it to all the client machines.
- **Clock adjustment:** The algorithm adjusts the clock of each client machine based on the difference between its local time and the time received from the time coordinator.
- **Average calculation:** The algorithm calculates the average time difference between the client machines and the time coordinator to reduce the effect of any clock drift.
- **Fault tolerance:** Berkeley's Algorithm is fault-tolerant, as it can handle failures in the network or the time coordinator by using backup time coordinators.
- **Accuracy: T**he algorithm provides accurate time synchronization across all the client machines, reducing the chances of errors due to time discrepancies.
- **Scalability:** The algorithm is scalable, as it can handle a large number of client machines, and the time coordinator can be easily replicated to provide high availability.
- **Security:** Berkeley's Algorithm provides security mechanisms such as authentication and encryption to protect the time information from unauthorized access or tampering.

Execution Steps:

Compile

Just make under dir "p1_berkeley_server_clients".

Server side

Run ./server under dir "p1_berkeley_server_clients".
In server.cpp I firstly setup normal socket and keep it listening.

Then there is a while loop which waiting for clients connections. When you open a new terminal window and run ./client it will catch connections from clients. At the bottom part of this while loop it asks you where you have enough clients, if yes this while loop will stop. Clients connection information will be stored into vectors client_sockets, client_ips and client_ports.
After confriming all clients have connected, then is the communication between serer and clients. The sever will send message to client asking for their lock clock value and sttore them into a vector clients_local_clocks. A for loop is used to iterate all client connections.
Then the server calculate the average value of all clock values, including itself, and how to adjust for each node. Then a for loop is used to send this adjustment offset to each client accordingly.

Then the server adjust itself's clock.

Client sides

Each time you run ./client under dir "p1_berkeley_server_clients" in a new opened terminal window, it creates a new client process.
In client.cpp it first connect to server whose ip and port is hard-coded. Then waiting for server's message.

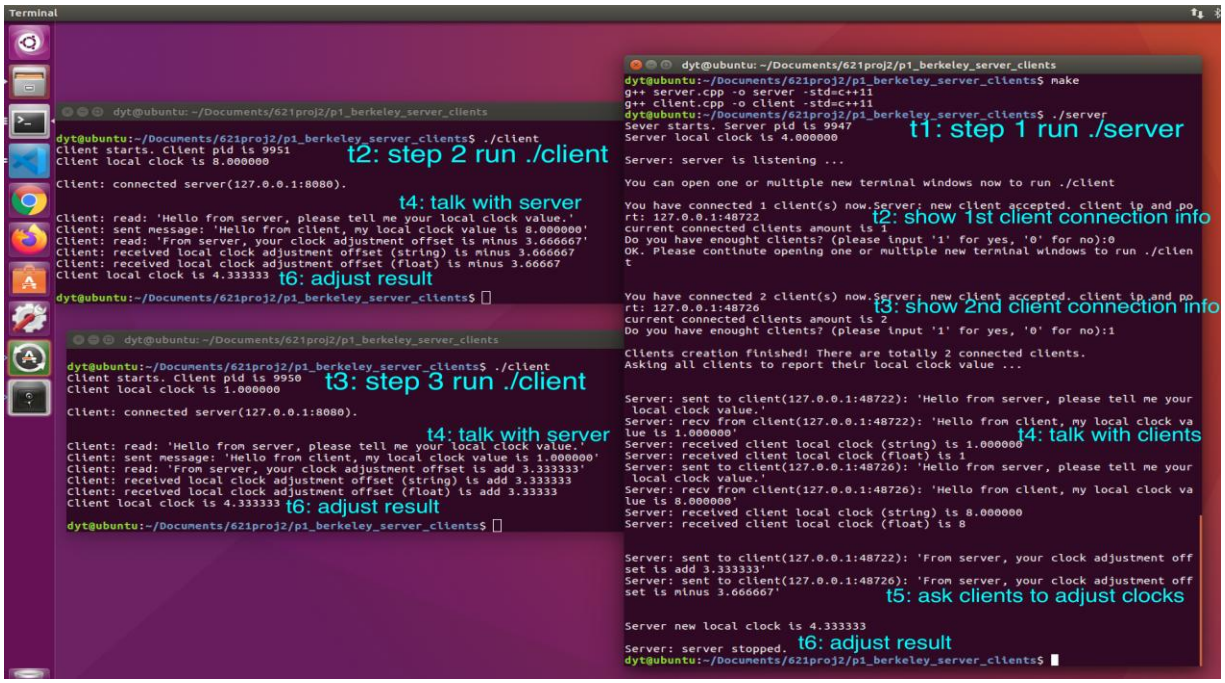
The first message it receives from server would be asking for its local clock value and the client will reply it. Then wait for new message from server.

The second message it receives from server would be how to asjust its clock and the client will do it. Then the client would have correct clock now.

At this moment, the server and all clients have clock synchronization.

The below screenshot shows how I ran it with 2 clients:

1. Run ./server in one terminal first.
2. Then run ./client in multiple ternimals.



Conclusion:

Thus we have studied Berkley's Algorithm.