DISTRIBUTED SYSTEMS PROJECT – 2 REPORT

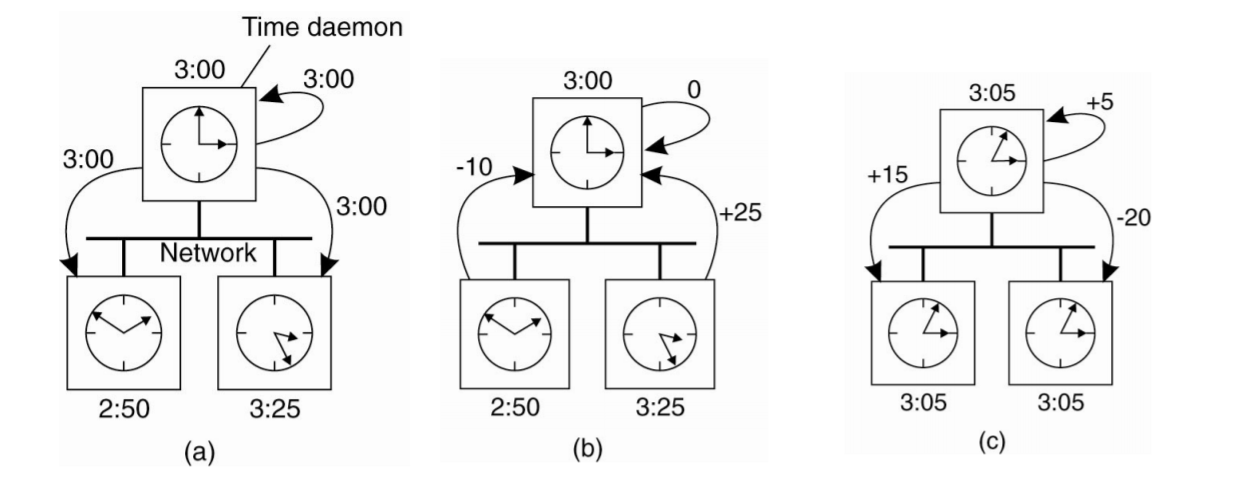
Project made by

Pramod Kampalli – 1001373669

Venkatesh Suvarna – 1001390986

***Implementation of Part 1 – Berkeley’s Clock Synchronization***

* We have implemented the Berkeley’s Algorithms by keeping the number of machines fixed(3), that is we have 2 clients and 1 server machine programs. We have used RMI(Remote Method Invocation) for communication between the programs.
* The Client programs are RMIClient1.java and RMIClient2.java while the interface program is in RMIInterface.java and the server program is RMIServer.java
* The working of our program is very much like the process given below



* Except that our program accepts whole integers as logical clock for the server(time daemon) and the two clients(as expected as the project)

***Implementation of Part 2 – Multicasting(Totally and Without Total)***

* We have implemented this part of the project in Python
* We have 3 client machines that can send any string messages to one another
* That is Client 1 can send messages to Client 2 and 3, Client 2 can send messages to Client 1 and 3, and Client 3 can send messages to Client 1 and 2
* Each client has a logical clock, that is initialized to 0 and then incremented by 1 when a client machine receives a message from any of the other two clients
* The client programs are self-explainatory

***Implementation of Part 3 – Distributed Locking(Mutual Exclusion)***

* We have implemented this part of the project in Python
* We have implemented the Distributed Locking mechanism using Lock Variables
* There are 3 client programs ie client1.py, client2.py and client3.py
* They all are trying to access a shared counter variable in the text file sharedfile.txt
* The value of a shared lock variable is in the text file lock.txt
* The value of the lock is binary that is it can have only two values – 0 or 1
* When the lock variable is set to 0, then the shared file is available for access, that is **no** process is currently accessing the sharedfile.txt
* When a process gains access to the sharedfile.txt, then it sets the shared lock variable to 1 – so that no other process tries to access sharedfile.txt
* When the lock variable is set to 1, then that means that a process has control over the sharedfile.txt and no other process should try to access it
* When a process has finished updating the counter value in sharedfile.txt, then it again sets the lock variable in lock.txt to 0, so that other clients can now try to access sharedfile.txt

***Learnings:***

* Berkeley’s algorithm is one of the easiest algorithms for logical clock synchronization, but the only disadvantage is that the time daemon becomes a bottleneck for the entire synchronization process
* Multicasting makes use of the message passing with timestamps, and Lamport logical clock fails at ensuring causality of events, which can be achieved using vector clocks.
* In totally ordered multicasting of messages, the order of the messages at all of the receiver machines is the same
* Also that lock variables is only one of the ways of achieving mutual exclusion between machines, the other methods include disabling interrupts, busy waiting, mutex etc.

***Challenges Faced:***

* Typecasting between byte and string during transfer of messages using socket in Python
* Handling multiple sockets per client machines for multicasting