YASH SUNIL NAREDI UID:1001405048 DATE:11/14/2017

“I haven’t given nor received unauthorized assistance on this work”

Algorithms used:

• Berkley’s Algorithm

• Lamport ‘s Algorithm

Assignments completed:

• Assignment-1

• Assignment-2

Technologies Used:

• Java (Multithreading) NETBEANS IDE

How to run:

1.Download JDK and NETBEANS from the website

2.Open the project on NETBEANS

3.Run Process0, Process1 and Process2 by opening each file individually and running them by clicking right click on each file or address in use error will be thrown.

4.

Note: It is necessary for all the processes to run at the same time for multithreading to occur or error will be thrown

My Learning:

**Berkley’s Algorithm**:

Server process in the Berkeley algorithm, called the master, periodically polls other slave processes. The algorithm is:

1.Let us assume that there are 3 clocks one master and two slaves

2.the master clock asks for the slave clocks for the time while asking itself also in the process

3. So for example the master clock has time 3:00, slave 1 has time 2:50 and slave 2 has time of 3:25, then the master clock will record the avg a -10 for slave 1 , 0 for self and +25 for slave 2 and take its average which is 5

4.So after the average is take the master clock updates its time by 5 mins and tells the other clock to update their time to 3:05, in this case telling slave 1 to adjust its time by +15mins and slave2 to reduce its time by 20 mins

This is the main concept of Berkley’s algorithm, now the algorithm states:

1.A master should be chosen via election like Robert’s algorithm

2. Master asks for slave timings like Cristian’s algorithm.

3. The master observes the round-trip time of the messages and estimates the time of each slave and its own.

4.The master then takes the average of clock time.

5. Instead of sending updates to the slaves about current time, the master sends the amount to be adjusted positive or negative to adjust its clock.

**Lamport’s Algorithm**

Requesting process:

1. Pushing its request in its own queue (ordered by time stamps)

2. Sending a request to every node.

3. Waiting for replies from all other nodes.

4. If own request is at the head of its queue and all replies have been received, enter critical section.

5. Upon exiting the critical section, remove its request from the queue and send a release message to every process.

Other processes:

1. After receiving a request, pushing the request in its own request queue (ordered by time stamps) and reply with a time stamp.

2. After receiving release message, remove the corresponding request from its own request queue.

3. If own request is at the head of its queue and all replies have been received, enter critical section.

1.After receiving a request, pushing the request in its own request queue (ordered by time stamps) and reply with a time stamp.

2.After receiving release message, remove the corresponding request from its own request queue.

3.If own request is at the head of its queue and all replies have been received, enter critical section.

How I implemented my program?

There are three processes:

• Process 0

• Process 1

• Process 2

Then there are three time nodes: (These classes are needed for sync of timing in the processes: Like the timer of Processes)

• Node0

• Node1

• Node2

What does the Program do?

Process0 acts as server to both Process1 and process 2

Process 1 is the server to Process 2 and for process 0

Process 2 is acting as a client for both for both.

The acknowledgements from the processes are named as ack0, ack1 and ack2.

Once the multicasting begins the timestamps are noted by each process and the time stamp matches.

What issues did I encounter?

If the ackhash (the token used in code to keep track of acknowledgements) is not empty or used before in the processes, then it disturbs the whole algorithm.

NETBEANS threw an error whenever I tried to run the program directly instead of clicking each file and running them separately

References:

•Youtube

•Wikipedia

•Lecture notes