Experiment 4

Aim : Implementation of Clock Synchronization

Theory:

Clock synchronization is a problem from computer science and engineering which deals with the idea that internal clocks of several computers may differ. Even when initially set accurately, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates.

Lamport's Algorithm

Each process maintains a single Lamport timestamp counter. Each event on the process is tagged with a timestamp from this counter. The counter is incremented before the event timestamp is assigned. If a process has four events, *a, b, c, d*, they would get Lamport timestamps of *1, 2, 3, 4*.

If an event is the sending of a message then the timestamp is sent along with the message. If an event is the receipt of a message then the the algorithm instructs you to compare the current value of the process' timestamp counter (which was just incremented before this event) with the timestamp in the received message.

If the timestamp of the received message is greater than that of the current system, the system timestamp is updated with that of the timestamp in the received message plus one. This ensures that the timestamp of the received event and all further timestamps will be greater than that of the timestamp of sending the message as well as all previous messages.

Lamport invented a simple mechanism by which the happened-before ordering can be captured numerically. A Lamport logical clock is an incrementing software counter maintained in each process.

It follows some simple rules:

1. A process increments its counter before each event in that process;
2. When a process sends a message, it includes its counter value with the message;
3. On receiving a message, the receiver process sets its counter to be the maximum of the message counter and its own counter incremented, before it considers the message received.

Program:

```
import java.io.*;
class Lamport
{
	public static void main(String args[]) throws IOException
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int p[],n,st,en,tlog[][],cnt[];
		String t[][];
		System.out.println("Enter the number of processes");
		n=Integer.parseInt(br.readLine());
		p=new int[n];
		tlog=new int[n][10];
		cnt=new int[n];
		t=new String[n][10];
```

```
                for(int i=0;i<n;i++)
                {
                        p[i]=0;
                        cnt[i]=0;
                }
                System.out.println("Enter the messages sent(0 0 to exit)");
                while(true)
                {
                        System.out.println("From process");
                        st=Integer.parseInt(br.readLine());
                        System.out.println("To process");
                        en=Integer.parseInt(br.readLine());
                        if(st==0 && en==0)
                                break;
                        p[st-1]+=1;
                        tlog[st-1][cnt[st-1]]=p[st-1];
                        t[st-1][cnt[st-1]++]="Send to process "+en;
                        p[en-1]=((p[en-1]+1)>(p[st-1]+1))?(p[en-1]+1):(p[st-1]+1);
                        tlog[en-1][cnt[en-1]]=p[en-1];
                        t[en-1][cnt[en-1]++]="Receive from process "+st;
                }
                for(int i=0;i<n;i++)
                {
                        System.out.println("\nFor process "+(i+1)+",the timestamps are\n");
                        System.out.println("Event\t\t\t\tTime\n");
                        for(int j=0;j<cnt[i];j++)
                                System.out.println(t[i][j]+"\t\t"+tlog[i][j]);
                }
        }
}
/*
Output :
C:\Users\Administrator\My Documents>javac Lamport.java
C:\Users\Administrator\My Documents>java Lamport
Enter the number of processes
3
Enter the messages sent(0 0 to exit)
From process
1
To process
2
From process
2
To process
3
From process
2
To process
1
From process
3
To process
```

```
1
From process
2
To process
3
From process
2
To process
1
From process
1
To process
3
From process
0
To process
0

For process 1,the timestamps are

Event                   Time

Send to process 2       1
Receive from process 2          5
Receive from process 3          6
Receive from process 2          7
Send to process 3       8


For process 2,the timestamps are

Event                   Time

Receive from process 1          2
Send to process 3       3
Send to process 1       4
Send to process 3       5
Send to process 1       6

For process 3,the timestamps are

Event                   Time

Receive from process 2          4
Send to process 1       5
Receive from process 2          6
Receive from process 1          9

*/
```

Experiment 5

Aim : Implementation of Election Algorithm (Bully/Ring Algorithm)

Theory:

Election Algorithms

 The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.

 An election algorithm is an algorithm for solving the coordinator election problem. By the nature of the coordinator election problem, any election algorithm must be a distributed algorithm.

-a group of processes on different machines need to choose a coordinator

-peer to peer communication: every process can send messages to every other process.

-Assume that processes have unique IDs, such that one is highest

-Assume that the priority of process Pi is i

(a) Bully Algorithm

Background: any process Pi sends a message to the current coordinator; if no response in T time units, Pi tries to elect itself as leader. Details follow:

Algorithm for process Pi that detected the lack of coordinator

1. Process Pi sends an “Election” message to every process with higher priority.

2. If no other process responds, process Pi starts the coordinator code running and sends a message to all processes with lower priorities saying “Elected Pi”

3. Else, Pi waits for T’ time units to hear from the new coordinator, and if there is no response start from step (1) again.

Algorithm for other processes (also called Pi)

If Pi is not the coordinator then Pi may receive either of these messages from Pj

if Pi sends “Elected Pj”; [this message is only received if $i < j$]

Pi updates its records to say that Pj is the coordinator

Else if Pj sends “election” message ($i > j$)

Pi sends a response to Pj saying it is alive

Pi starts an election.

Programs