**ASSIGNMENT NO: 9**

**Title:**

Lamport’s Logical Clock

**Problem Statement:**

Implement Lamport’s Logical Clock algorithm for clock synchronization.

**Tools/Environment:**

The following tools and technologies will be used in this project:

Programming language: Java

IDE: Eclipse

Networking: Java Sockets

**Theory:**

**Logical Clocks**refer to implementing a protocol on all machines within your distributed system so that the machines are able to maintain the consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows the global ordering of events from different processes in such systems.

**Implementation:**

Step 1: Install the required tools and libraries, including Java, and Eclipse.

Step 2: Create a simulated distributed system using Mininet. This can be done by creating multiple nodes connected by a network.

Step 3: Designate one of the nodes as the time server.

Step 4: Implement the algorithm in Java. This can be done by writing a program that performs the following steps:

* Step 1: Each process in the system should have a Lamport clock initialized to a known value. The value of the clock can be any integer, but it is usually set to 0.
* Step 2: When a process performs an internal event, it should increment its Lamport clock using the tick() operation. This operation increments the clock's value by 1.
* Step 3: When a process sends a message, it should include its Lamport clock's current value in the message. Before sending the message, the process should increment its Lamport clock using the sendEvent() operation. This operation increments the clock's value by 1.
* Step 4: When a process receives a message, it should update its Lamport clock's value using the receiveEvent() operation. This operation takes as input the timestamp included in the message and updates the clock's value to be the maximum of its current value and the received timestamp plus 1.
* Step 5: After performing any operation (internal or external), a process can obtain the current value of its Lamport clock using the getValue() operation.
* Step 5: Run the program and observe the performance of the algorithm in terms of synchronization accuracy and overhead.

**Conclusion:**

The algorithm allows each process in the system to maintain a Lamport clock that can be used to timestamp events, such as message sends and receives.

By using Lamport's Logical Clock algorithm, processes can ensure that they maintain a consistent global ordering of events, even in the presence of message delays or reordering. The algorithm is simple to implement and can be used in a wide range of distributed systems.

**Code:**

Server-Side Code:

import java.net.\*;

import java.io.\*;

public class Server {

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(8086);

System.*out*.println("Server started");

int counter = 0;

while (true) {

Socket clientSocket = serverSocket.accept();

System.*out*.println("Client connected: " + clientSocket.getInetAddress().getHostName());

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

String inputLine;

while ((inputLine = in.readLine()) != null) {

System.*out*.println("Received from client: " + inputLine);

String[] parts = inputLine.split(" ");

int eventTime = Integer.*parseInt*(parts[0]);

String message = parts[1];

counter = Math.*max*(counter, eventTime) + 1; // update the counter

String response = counter + " ACK " + message;

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

out.println(response);

}

clientSocket.close();

System.*out*.println("Client disconnected");

}

} catch (IOException e) {

System.*out*.println("Error: " + e.getMessage());

}

}

}

Client-Side code:

import java.net.\*;

import java.io.\*;

import java.util.Random;

public class Client {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 8086);

System.*out*.println("Connected to server: " + socket.getInetAddress().getHostName());

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

int eventTime = 0;

Random rand = new Random();

for (int i = 0; i < 5; i++) {

Thread.*sleep*(rand.nextInt(1000)); // simulate a random event

eventTime++;

String message = "Message " + i;

String request = eventTime + " " + message;

out.println(request);

String response = in.readLine();

String[] parts = response.split(" ");

int counter = Integer.*parseInt*(parts[0]);

int ackTime = counter;

String ackMessage = parts[2];

System.*out*.println("Received ACK for message to Tejas Kadam " + i + " with counter " + ackTime + " and message " + ackMessage);

}

socket.close();

System.*out*.println("Disconnected from server");

} catch (IOException | InterruptedException e) {

System.*out*.println("Error: " + e.getMessage());

}

}

}

**Output:**

