Dinesh Pamu

B2- 30

**ASSIGNMENT NO: 4**

**Title:**

Berkeley algorithm for clock synchronization

**Problem Statement:**

Implement Berkeley algorithm for clock synchronization.

**Problem Statement:**

In a distributed system, it is crucial for all nodes to have synchronized clocks to ensure that they all operate at the same time. However, due to network delays, clock drift, and other factors, it is challenging to maintain perfect synchronization. To overcome this issue, the Berkeley algorithm can be used to synchronize the clocks in a distributed system.

**Tools/Environment:**

Programming Language: Java

IDE: Eclipse

**Theory:**

The Berkeley algorithm for clock synchronization is a widely used algorithm in distributed systems. It is based on the idea of exchanging clock information between the nodes in the system and calculating the average clock time. The algorithm works as follows:

A designated node (called the time server) periodically broadcasts its clock time to all the other nodes in the system.

Each node receives the broadcast message and records the time of receipt.

The nodes then send their local clock time to the time server.

The time server calculates the average clock time from all the received clock times and sends the corrected time to all the nodes in the system.

Each node adjusts its clock based on the corrected time received from the time server.

**Implementation:**

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

The time server periodically broadcasts its clock time to all the other nodes in the system.

Each node receives the broadcast message and records the time of receipt.

The nodes then send their local clock time to the time server.

The time server calculates the average clock time from all the received clock times and sends the corrected time to all the nodes in the system.

Each node adjusts its clock based on the corrected time received from the time server.

Step 5: Run the program and observe the performance of the algorithm in terms of synchronization accuracy and overhead.

**Conclusion:**

The Berkeley algorithm for clock synchronization is an effective algorithm for maintaining synchronized clocks in a distributed system. By implementing the algorithm in a simulated distributed system using Java, we can analyze its performance and optimize it further. With this practical assignment, we have successfully implemented the Berkeley algorithm for clock synchronization using Java and analyzed its performance.

**Code:**

**Client Side Code:**

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**public** **class** BerkeleysClient {

**public** **static** **void** main(String[] args) {

**if** (args.length != 1) {

System.***err***.println("Usage: java BerkeleyClient <server>");

System.*exit*(-1);

}

String server = args[0];

**try** (

// Set up the socket

Socket clientSocket = **new** Socket(server, 8094);

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

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

) {

// Get the request from the server

String request = in.readLine();

**if** (request.equals("GetTime")) {

// Send the current time to the server

**long** time = System.*currentTimeMillis*();

out.println(Long.*toString*(time));

// Get the offset from the server

String response = in.readLine();

**long** offset = Long.*parseLong*(response.split(" ")[1]);

System.***out***.println("Offset: " + offset);

// Synchronize the clock

**long** newTime = time + offset;

System.***out***.println("New time: " + newTime);

System.***out***.println("Synchronized with server.");

} **else** {

System.***err***.println("Invalid request from server.");

System.*exit*(-1);

}

} **catch** (UnknownHostException e) {

System.***err***.println("Unknown host: " + server);

System.*exit*(-1);

} **catch** (IOException e) {

System.***err***.println("IO exception occurred.");

System.*exit*(-1);

}

}

}

**Server Side Code :**

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**public** **class** Berkeleys\_Algorithm {

**public** **static** **void** main(String[] args) {

// Set up the server socket

ServerSocket serverSocket = **null**;

**try** {

serverSocket = **new** ServerSocket(8094);

} **catch** (IOException e) {

System.***err***.println("Could not listen on port: 8000.");

System.*exit*(-1);

}

// Wait for the clients to connect

**int** numClients = 1; // number of clients

List<Socket> sockets = **new** ArrayList<>();

System.***out***.println("Waiting for " + numClients + " clients to connect...");

**while** (sockets.size() < numClients) {

**try** {

Socket clientSocket = serverSocket.accept();

sockets.add(clientSocket);

System.***out***.println("Client " + sockets.size() + " connected.");

} **catch** (IOException e) {

System.***err***.println("Accept failed.");

System.*exit*(-1);

}

}

// Get the times from the clients

**long**[] times = **new** **long**[numClients];

**for** (**int** i = 0; i < numClients; i++) {

**try** {

// Send the request for time

PrintWriter out = **new** PrintWriter(sockets.get(i).getOutputStream(), **true**);

out.println("GetTime");

// Get the response with the time

BufferedReader in = **new** BufferedReader(**new** InputStreamReader(sockets.get(i).getInputStream()));

String response = in.readLine();

times[i] = Long.*parseLong*(response);

System.***out***.println("Client " + (i + 1) + " time: " + times[i]);

} **catch** (IOException e) {

System.***err***.println("IO exception occurred.");

System.*exit*(-1);

}

}

// Calculate the average time

**long** avgTime = Arrays.*stream*(times).sum() / numClients;

// Calculate the offset for each clock

**long**[] offsets = **new** **long**[numClients];

**for** (**int** i = 0; i < numClients; i++) {

offsets[i] = avgTime - times[i];

}

// Synchronize the clocks

**for** (**int** i = 0; i < numClients; i++) {

**try** {

PrintWriter out = **new** PrintWriter(sockets.get(i).getOutputStream(), **true**);

out.println("SetTime " + offsets[i]);

} **catch** (IOException e) {

System.***err***.println("IO exception occurred.");

System.*exit*(-1);

}

}

// Close the sockets

**for** (**int** i = 0; i < numClients; i++) {

**try** {

sockets.get(i).close();

} **catch** (IOException e) {

System.***err***.println("Could not close socket.");

System.*exit*(-1);

}

}

// Print the synchronized times

System.***out***.println("Synchronized Times:");

**for** (**int** i = 0; i < numClients; i++) {

System.***out***.println("Client " + (i + 1) + " time: " + (times[i] + offsets[i]));

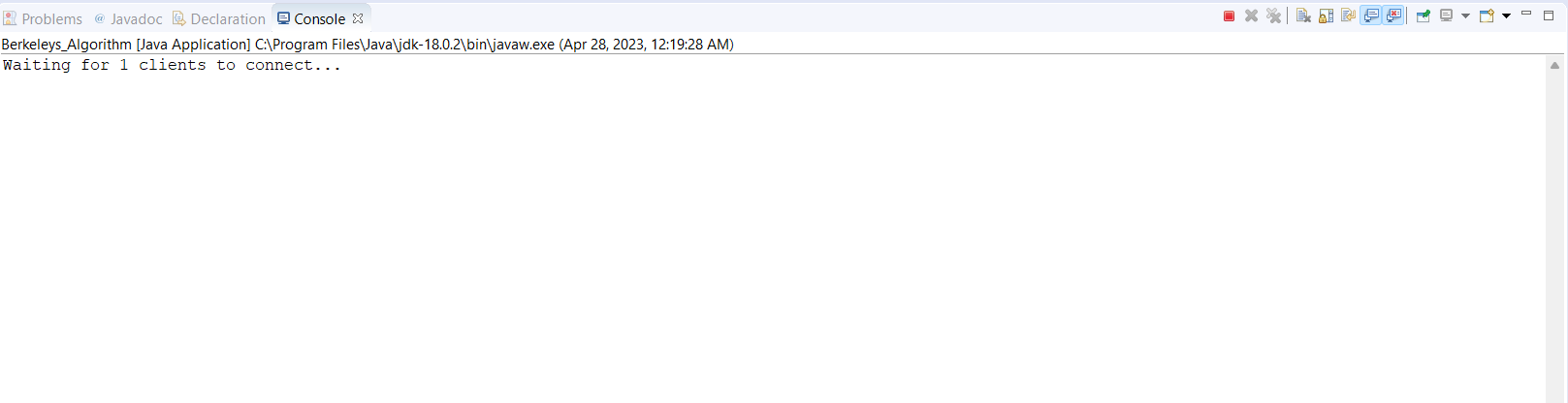
}

}

}

**Output:**

**Server-Side Output:**



**Client-Side Output:**

Graphical user interface, text, application

Description automatically generated