

Experiment No: 4

Aim: Implementation of Clock synchronization i.e. Berkeley Algorithm

Theory:

Two clocks are said to be synchronized at a particular instance of time if the clock skew of the two clocks is less than some specified constant δ . A set of clocks are said to be synchronized if the clock skew of any two clocks in this set is less than δ .

Clock Synchronization Issues:

- A distributed system requires:
 1. External Synchronization: Synchronize with an external time source
 2. Internal Synchronization: Clocks within the same network synchronize with each other
- Clock Synchronization requires:
 1. Each node can read the other nodes' clock value: Must consider unpredicted communication delay
 2. Time must never run backward: Smooth adjustments \Rightarrow must maintain the order of the events
- Clock Synchronization Algorithms:
 1. Centralized
 - Passive Time Server Centralized Algorithm
 - Cristian's algorithm
 - Active Time Server Centralized Algorithm
 - Berkeley Algorithm
 2. Distributed
 - Global Averaging Distributed Algorithms
 - Localized Averaging Distributed Algorithms

Berkeley Algorithm:

1. While Cristian's algorithm is passive, Berkeley UNIX takes an opposite approach.
2. The time server will poll every machine periodically to ask what time it is there.

3. Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks until some specified reduction has been achieved.
4. This method is suitable for a system in which no machine has a WWV receiver getting the UTC.
5. The time daemon's time must be set manually by the operator periodically.

Code:

Client

```
1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStreamReader;
4 import java.io.PrintWriter;
5 import java.net.Socket;
6 import java.text.ParseException;
7 import java.text.SimpleDateFormat;
8 import java.time.LocalDateTime;
9 import java.time.format.DateTimeFormatter;
10 import java.util.Date;
11 public class Client {
12     private static void startSendingTime(Socket slaveSocket) throws IOException, InterruptedException {
13         PrintWriter out = new PrintWriter(slaveSocket.getOutputStream(), true);
14         while (true) {
15             out.println(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date()));
16             System.out.println("Recent time sent successfully\n");
17             Thread.sleep(5000);
18         }
19     }
20     private static void startReceivingTime(Socket slaveSocket) throws IOException, ParseException {
21         BufferedReader in = new BufferedReader(new InputStreamReader(slaveSocket.getInputStream()));
22         while (true) {
23             String receivedTimeStr = in.readLine();
24             receivedTimeStr = receivedTimeStr.trim();
25             LocalDateTime synchronizedTime = LocalDateTime.parse(receivedTimeStr, DateTimeFormatter.ISO_LOCAL_DATE_TIME);
26             System.out.println("Synchronized time at the client is: " + synchronizedTime + "\n");
27         }
28     }
29     private static void initiateSlaveClient(int port) throws IOException {
30         Socket slaveSocket = new Socket("127.0.0.1", port);
31         System.out.println("Starting to receive time from server\n");
32         Thread sendTimeThread = new Thread(() -> {
33             try {
34                 startSendingTime(slaveSocket);
35             } catch (IOException | InterruptedException e) {
36                 e.printStackTrace();
37             }
38         });
39         sendTimeThread.start();
40         System.out.println("Starting to receive synchronized time from server\n");
41         Thread receiveTimeThread = new Thread(() -> {
42             try {
43                 startReceivingTime(slaveSocket);
44             } catch (IOException | ParseException e) {
45                 e.printStackTrace();
46             }
47         });
48         receiveTimeThread.start();
49     }
50     public static void main(String[] args) throws IOException {
51         initiateSlaveClient(8080);
52     }
53 }
54
```

Server

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.HashMap;
import java.util.Map;

public class ClockServer {

    private static final Map<String, ClientData> clientData = new HashMap<>();
    private static void startReceivingClockTime(Socket masterSlaveConnector,
String address)
        throws IOException, InterruptedException {
        BufferedReader in = new BufferedReader(new
InputStreamReader(masterSlaveConnector.getInputStream()));
        while (true) {
            String clockTimeString = in.readLine();
            clockTimeString = clockTimeString.trim();
            LocalDateTime clockTime = LocalDateTime.parse(clockTimeString,
                DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
            Duration timeDifference = Duration.between(clockTime,
LocalDateTime.now());
            clientData.put(address, new ClientData(clockTime, timeDifference,
masterSlaveConnector));
            System.out.println("Client Data updated with: " + address + "\n");
            Thread.sleep(5000);
        }
    }

    private static void startConnecting(ServerSocket masterServer) throws
IOException {
        while (true) {
            Socket masterSlaveConnector = masterServer.accept();
            String slaveAddress =
masterSlaveConnector.getInetAddress().getHostAddress() + ":"
                + masterSlaveConnector.getPort();

            System.out.println(slaveAddress + " got connected successfully");
            Thread currentThread = new Thread(() -> {
                try {
                    startReceivingClockTime(masterSlaveConnector,
slaveAddress);
```

```

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    });
    currentThread.start();
}
}

private static Duration getAverageClockDiff() {
    Map<String, ClientData> currentClientData = new HashMap<>(clientData);

    Duration sumOfClockDifference = currentClientData.values().stream()
        .map(ClientData::getTimeDifference)
        .reduce(Duration.ZERO, Duration::plus);

    return sumOfClockDifference.dividedBy(currentClientData.size());
}

private static void synchronizeAllClocks() throws IOException {
    while (true) {
        System.out.println("New synchronization cycle started.");
        System.out.println("Number of clients to be synchronized: " +
clientData.size());
        if (!clientData.isEmpty()) {
            Duration averageClockDifference = getAverageClockDiff();
            for (Map.Entry<String, ClientData> entry :
clientData.entrySet()) {
                try {
                    LocalDateTime synchronizedTime =
LocalDateTime.now().plus(averageClockDifference);
                    entry.getValue().getConnector().getOutputStream().writ
e((synchronizedTime + "\n").getBytes());
                } catch (Exception e) {
                    System.out.println(
                        "Something went wrong while sending
synchronized time through " + entry.getKey());
                }
            }
        } else {
            System.out.println("No client data. Synchronization not
applicable.");
        }

        System.out.println("\n\n");
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

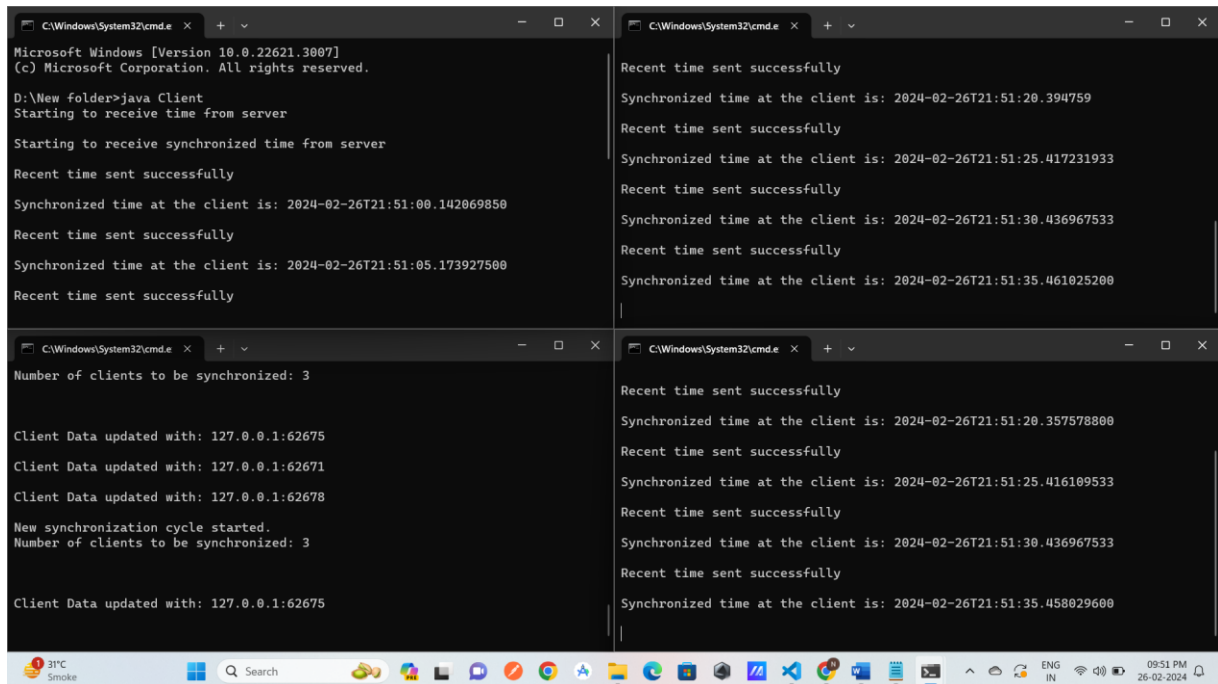
```

```

    }
    private static void initiateClockServer(int port) throws IOException {
        ServerSocket masterServer = new ServerSocket(port);
        System.out.println("Socket at master node created successfully\n");
        System.out.println("Clock server started...\n");
        System.out.println("Starting to make connections...\n");
        Thread masterThread = new Thread(() -> {
            try {
                startConnecting(masterServer);
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        masterThread.start();
        System.out.println("Starting synchronization parallelly...\n");
        Thread syncThread = new Thread(() -> {
            try {
                synchronizeAllClocks();
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
        syncThread.start();
    }
    public static void main(String[] args) throws IOException {
        initiateClockServer(8080);
    }
    private static class ClientData {
        private final LocalDateTime clockTime;
        private final Duration timeDifference;
        private final Socket connector;
        public ClientData(LocalDateTime clockTime, Duration timeDifference,
Socket connector) {
            this.clockTime = clockTime;
            this.timeDifference = timeDifference;
            this.connector = connector;
        }
        public LocalDateTime getClockTime() {
            return clockTime;
        }
        public Duration getTimeDifference() {
            return timeDifference;
        }
        public Socket getConnector() {
            return connector;
        }
    }
}

```

Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.3807]
(c) Microsoft Corporation. All rights reserved.

D:\New Folder>java Client
Starting to receive time from server

Starting to receive synchronized time from server
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:00.142069850
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:05.173927500
Recent time sent successfully

C:\Windows\System32\cmd.exe
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:20.394759
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:25.417231933
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:30.436967533
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:35.461025200

C:\Windows\System32\cmd.exe
Number of clients to be synchronized: 3

Client Data updated with: 127.0.0.1:62675
Client Data updated with: 127.0.0.1:62671
Client Data updated with: 127.0.0.1:62678
New synchronization cycle started.
Number of clients to be synchronized: 3

Client Data updated with: 127.0.0.1:62675

C:\Windows\System32\cmd.exe
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:20.357578800
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:25.416109533
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:30.436967533
Recent time sent successfully
Synchronized time at the client is: 2024-02-26T21:51:35.458029600
```

Conclusion:

In a centralized system the solution is trivial; the centralized server will dictate the system time. Cristian's algorithm and the Berkeley Algorithm are some solutions to the clock synchronization problem in a centralized server environment. In a distributed system the problem takes on more complexity because a global time is not easily known.