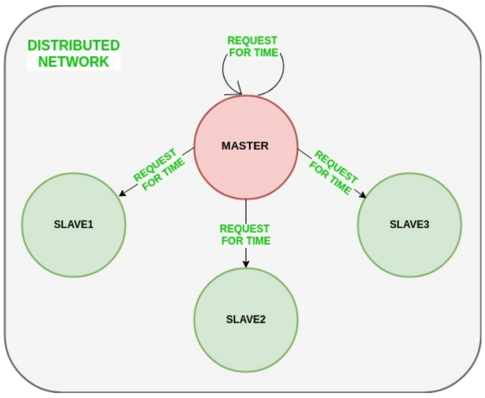
Distributed Computing

Experiment – 4

Name: Om Ghate **Roll No: 23** **Batch: A**

Program to demonstrate Clock Synchronization	
Learning Objective:	Implement techniques for clock synchronization.
Learning Outcome:	Ability to demonstrate Clock Synchronization.
Course Outcome:	**CSL801.2**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO**3) **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. **(PO5)Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
Bloom's Taxonomy Level:	Analysis, Create
Theory:	Clock synchronization in distributed systems aims to establish a reference for time across nodes. Imagine a scenario where three distinct systems are part of a distributed environment. In order for data exchange and coordinated operations to take place it is essential that these systems have a shared understanding of time. Achieving clock synchronization ensures that data flows seamlessly between them tasks are executed coherently and communication happens without any ambiguity.



Lamport's Logical Clocks:
Lamport's algorithm is a logical clock synchronization method that assigns a timestamp to events in a distributed system. It uses a partial order on events and ensures that events occurring in a certain order receive unique timestamps. The logical clocks are not tied to real time but provide a consistent ordering of events, aiding in establishing a causal relationship between them. Lamport clocks are incremented with each event and are compared to determine their relative order.

Berkley Algorithm:
The Berkley algorithm is a clock synchronization protocol designed to synchronize the clocks of machines in a distributed system. It involves a centralized time server, which periodically broadcasts its time to the other machines. The client machines adjust their local clocks based on the received time and the round-trip delays. Berkley aims to minimize clock skew among the machines in the network, providing a more synchronized sense of time across the distributed environment. This helps in maintaining temporal consistency for various applications and processes running on different nodes.

Types of Clock Synchronization

1. Physical clock synchronization
2. Logical clock synchronization
3. Mutual exclusion synchronization

1. Physical clock synchronization

Physical clock synchronization is a method used to maintain harmony in distributed systems by aligning clocks with Universal Coordinated Time (UTC), a recognized standard. This helps to minimize time differences between nodes and ensures that all nodes have a consistent clock. By adjusting clocks to UTC, diverse

	systems can work together, ensuring a consistent and accurate timekeeping system. This approach is crucial for ensuring the smooth functioning of distributed systems. **2. Logical clock synchronization** Logical clock synchronization focuses on establishing connections between events, enhancing comprehension and coordination. This approach to understanding behavior is akin to a choreographed dance, where steps are sequenced for execution. In distributed systems, absolute time often takes a backseat to clock synchronization, which prioritizes the order of events over their exact timing. Thus, clock synchronization plays a crucial role in managing events in distributed systems. **3. Mutual exclusion synchronization** Mutual exclusion synchronization is an expert technique that promotes resource harmony by ensuring different processes take turns accessing shared resources. This approach helps avoid conflicts and collisions, ensuring efficient resource utilization and overall system efficiency. By synchronizing swimmers, distributed systems ensure resources are accessed sequentially, minimizing conflicts and collisions. This synchronization approach enhances resource utilization and overall system efficiency, making distributed systems a dynamic and efficient ecosystem.
Algorithm :	Berkeley Algorithm *1) An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.* *2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.* *3) Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.*

Programing Code -	Programs -
	
	ClockClient.java
	
	import java.io.*;
	import java.net.*;
	
	public class ClockClient {

```
    public static void main(String[] args) {
        final String serverName = "localhost";
        final int port = 9090;

        try {
            Socket socket = new Socket(serverName, port);
            System.out.println("Connected to server.");

            // Get input and output streams
            OutputStream outToServer = socket.getOutputStream();
            InputStream inFromServer = socket.getInputStream();

            // Get client's current time
            long clientTime = System.currentTimeMillis();
            System.out.println("Client time: " + clientTime);

            // Send client's time to server
            DataOutputStream out = new DataOutputStream(outToServer);
            out.writeLong(clientTime);

            // Receive server's time
            DataInputStream in = new DataInputStream(inFromServer);
            long serverTime = in.readLong();
            System.out.println("Server time: " + serverTime);

            // Calculate time difference
            long timeDifference = serverTime - clientTime;
            System.out.println("Time difference: " + timeDifference);

            // Adjust client's clock
            long adjustedTime = System.currentTimeMillis() + timeDifference;
            System.out.println("Adjusted time: " + adjustedTime);

            // Close the connection
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ClockServer.java

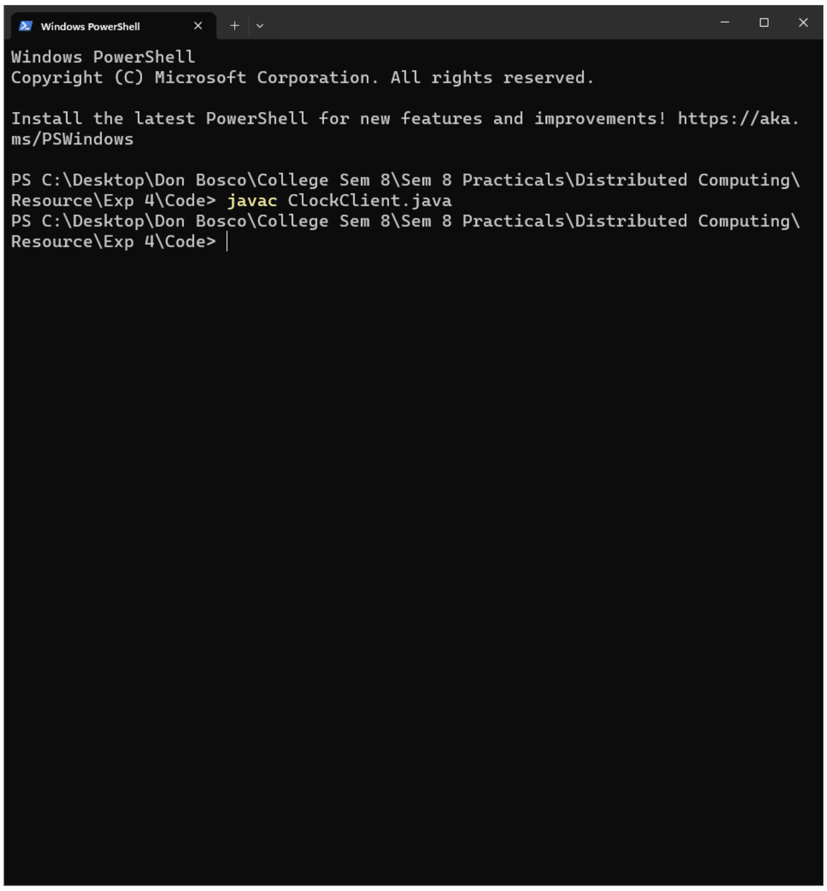
```
import java.io.*;
import java.net.*;

public class ClockServer {
    public static void main(String[] args) {
        final int port = 9090;

        try {
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server started. Waiting for clients...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " + clientSocket.getInetAddress());

                // Handle client in a separate thread
                Thread clientThread = new Thread(new ClientHandler(clientSocket));
                clientThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler implements Runnable {
    private final Socket clientSocket;

    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            // Get input and output streams
            InputStream inFromClient = clientSocket.getInputStream();
            OutputStream outToClient = clientSocket.getOutputStream();

            // Receive client's time
            DataInputStream in = new DataInputStream(inFromClient);
            long clientTime = in.readLong();
```

```java
            System.out.println("Received client time: " + clientTime);

            // Get server's current time
            long serverTime = System.currentTimeMillis();
            System.out.println("Server time: " + serverTime);

            // Send server's time to client
            DataOutputStream out = new DataOutputStream(outToClient);
            out.writeLong(serverTime);

            // Close the connection
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output

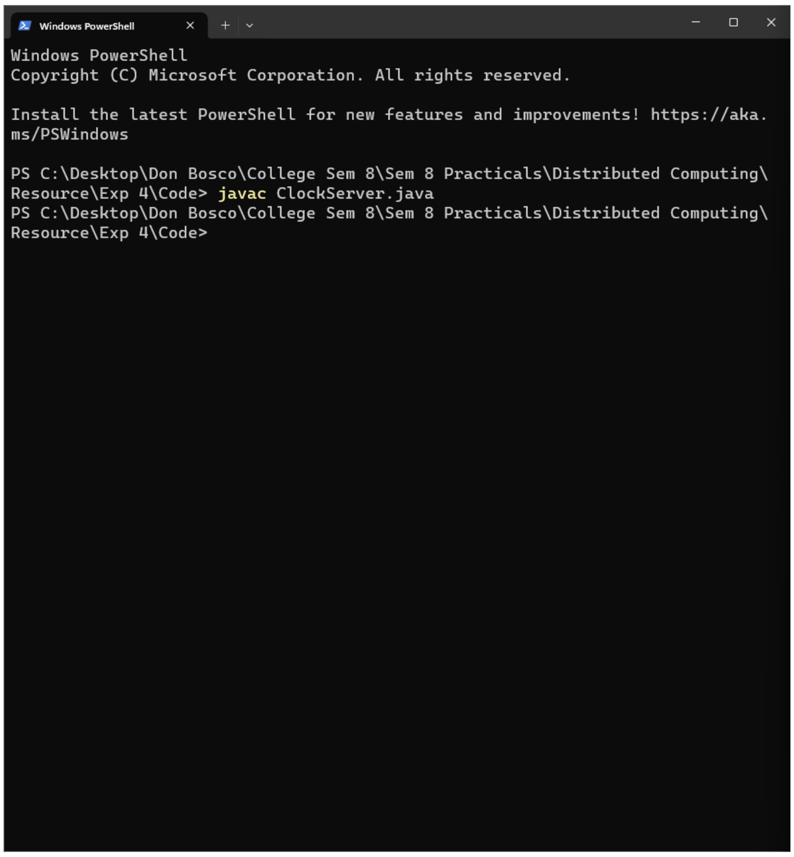
Compilation -

ClockClient.java



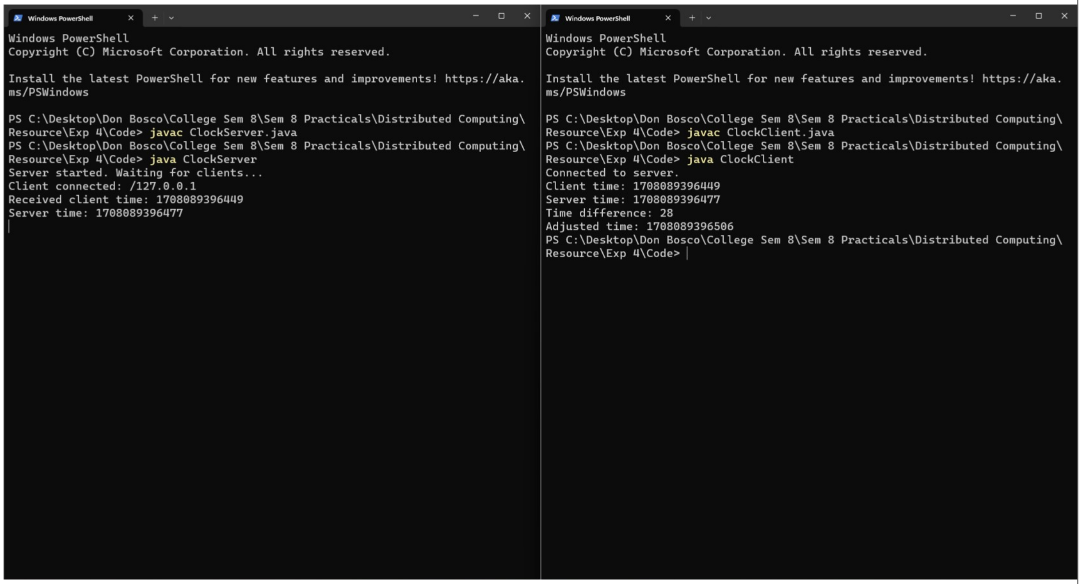
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockClient.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> |
```

ClockServer.java



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockServer.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code>
```

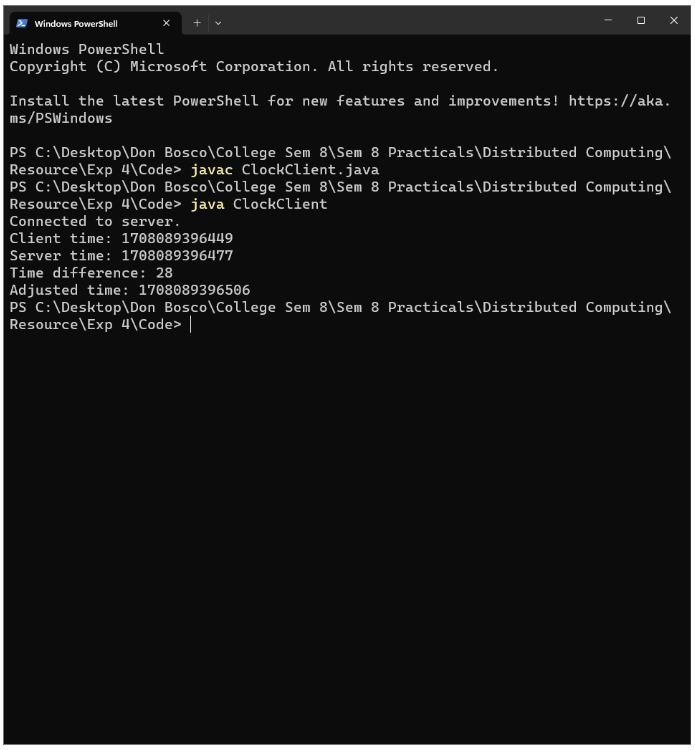
Implementation -



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockServer.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> java ClockServer
Server started. Waiting for clients...
Client connected: /127.0.0.1
Received client time: 1708089396449
Server time: 1708089396477
```

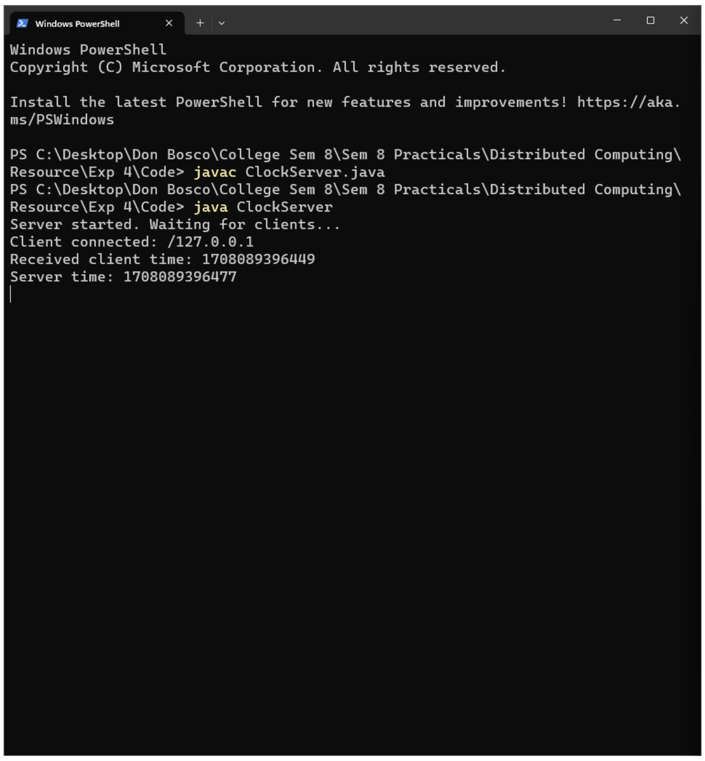
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockClient.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> java ClockClient
Connected to server.
Client time: 1708089396449
Server time: 1708089396477
Time difference: 28
Adjusted time: 1708089396506
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code>
```

ClockClient.java



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockClient.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> java ClockClient
Connected to server.
Client time: 1708089396449
Server time: 1708089396477
Time difference: 28
Adjusted time: 1708089396506
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code>
```

ClockServer.java



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.
ms/PSWindows

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> javac ClockServer.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\
Resource\Exp 4\Code> java ClockServer
Server started. Waiting for clients...
Client connected: /127.0.0.1
Received client time: 1708089396449
Server time: 1708089396477
```

Conclusion :	Thus, we have studied basic underlying concepts of clock synchronization and successfully implemented Berkeley’s Algorithm for clock synchronization in distributed systems.
References:	https://www.geeksforgeeks.org/berkeleys-algorithm/ https://www.geeksforgeeks.org/clock-synchronization-in-distributed-system/ moodle notes

Rubrics for Assessment

Timely Submission	Submitted after 2 weeks 0	Submitted after deadline 1	On time Submission 2
Understanding	Student is confused about the concept 0	Students has justifiably understood the concept 2	Students is very clear about the concepts 3
Performance	Students has not performed the Experiment 0	Student has performed with help 2	Student has independently performed the experiment 3
Development	Student struggles to write code 0	Student can write code the requirement stated 1	Student can write exceptional code with his own ideas 2