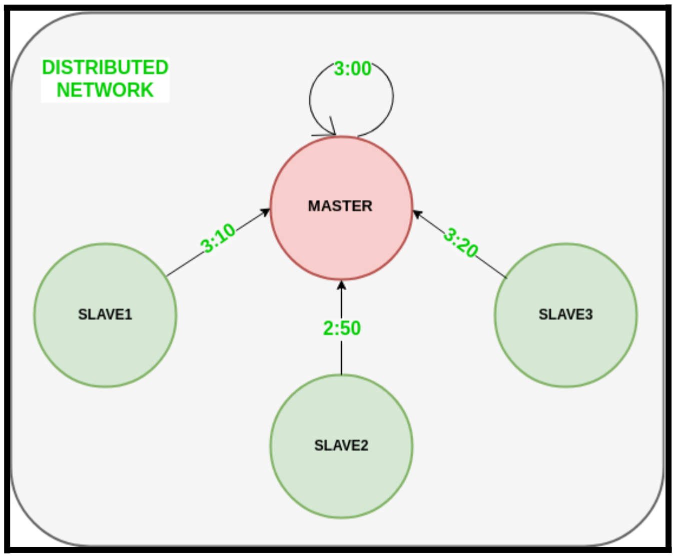
EXPERIMENT 5

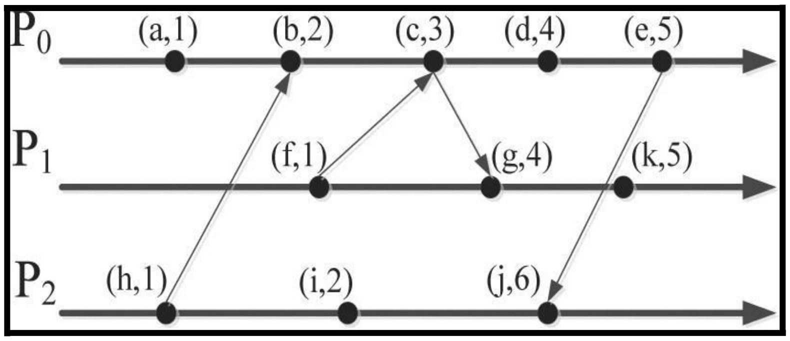
Aim: Implementation a Clock Synchronization algorithms

Theory:

A Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes.

Clock synchronization algorithms are used to ensure that the clocks in a distributed system are approximately synchronized, meaning they show a similar time across different nodes or devices. These algorithms help in maintaining consistency in time measurements, enabling various distributed applications and systems to operate effectively. They typically involve protocols or mechanisms to adjust the local clocks based on reference time sources, such as time servers or other nodes in the network.





Types of Clock Synchronization :

1. **Physical clock synchronization :**
   In the realm of distributed systems each node operates with its clock, which can lead to time differences. However the goal of physical clock synchronization is to overcome this challenge.

2. **Logical clock synchronization :**
   Within the tapestry of distributed systems absolute time often takes a backseat to clock synchronization. Think of clocks as storytellers that prioritize the order of events than their exact timing..

3. **Mutual exclusion synchronization :**
   In the bustling symphony of distributed systems one major challenge is managing shared resources. Imagine multiple processes competing for access to the resource simultaneously.

Working of Clock Synchronization Algorithms:

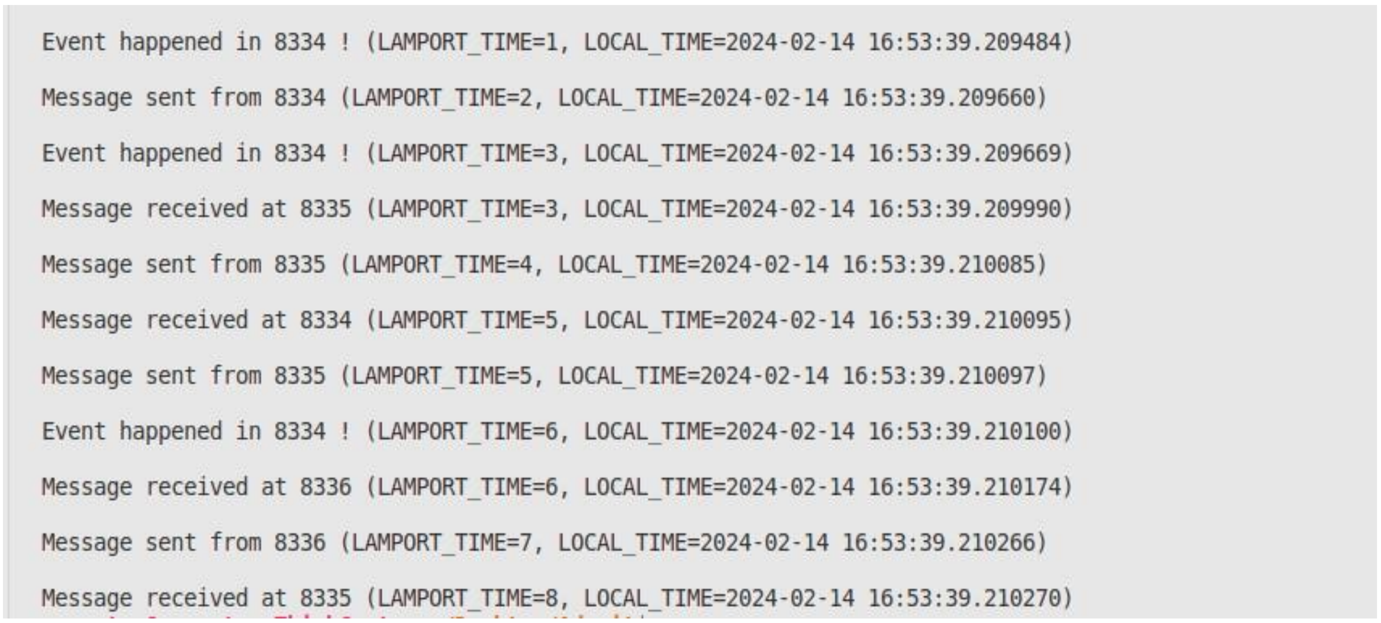
- Clock synchronization algorithms employ various techniques to adjust the local clocks of different nodes or devices within a distributed system.
- These algorithms often utilize a reference time source, such as a time server or a master clock, to provide a basis for synchronization.
- Nodes periodically exchange time information or synchronize their clocks with the reference time source using predefined protocols or synchronization mechanisms.
- Clock adjustments are made based on the time differences observed between the local clock and the reference time source, accounting for factors such as network delays and clock drift.
- By synchronizing the clocks across the distributed system, these algorithms help in achieving consistent and accurate time measurements for various operations and events.
- Clock synchronization is crucial for ensuring the proper functioning of distributed applications, including distributed databases, communication protocols, and distributed computing systems.
- Common clock synchronization algorithms include NTP (Network Time Protocol), PTP (Precision Time Protocol), and various variations of clock synchronization techniques tailored to specific requirements and constraints of different distributed systems.
- The ultimate goal of clock synchronization algorithms is to maintain time consistency and accuracy across distributed nodes, facilitating efficient coordination and operation of distributed systems and applications.

Code:

```
class MyThread extends Thread {
public void run() {
try {
synchronized (System.out) {
System.out.println("Thread " + Thread.currentThread().getId() + " is running");
}
Thread.sleep(1000);
synchronized (System.out) {
System.out.println("Thread " + Thread.currentThread().getId() + " has completed");
}
} catch (InterruptedException e) {
synchronized (System.out) {
System.out.println("Thread interrupted");
}}}}
class MyRunnable implements Runnable {
public void run() {
try {
synchronized (System.out) {
System.out.println("Runnable Thread " + Thread.currentThread().getId() + " is
running");
}
Thread.sleep(1000);
synchronized (System.out) {
System.out.println("Runnable Thread " + Thread.currentThread().getId() + " has
completed");
}
} catch (InterruptedException e) {
synchronized (System.out) {
System.out.println("Runnable Thread interrupted");
}}}}
public class MultithreadingExample {
public static void main(String[] args) {
int n = 5; // Number of threads
System.out.println("Using Thread class:");
for (int i = 0; i < n; i++) {
MyThread thread = new MyThread();
thread.start();
}
System.out.println("\nUsing Runnable interface:");
for (int i = 0; i < n; i++) {
```

```
Thread thread = new Thread(new MyRunnable());
thread.start();
}}}
```

Output:



```
Event happened in 8334 ! (LAMPORT_TIME=1, LOCAL_TIME=2024-02-14 16:53:39.209484)

Message sent from 8334 (LAMPORT_TIME=2, LOCAL_TIME=2024-02-14 16:53:39.209660)

Event happened in 8334 ! (LAMPORT_TIME=3, LOCAL_TIME=2024-02-14 16:53:39.209669)

Message received at 8335 (LAMPORT_TIME=3, LOCAL_TIME=2024-02-14 16:53:39.209990)

Message sent from 8335 (LAMPORT_TIME=4, LOCAL_TIME=2024-02-14 16:53:39.210085)

Message received at 8334 (LAMPORT_TIME=5, LOCAL_TIME=2024-02-14 16:53:39.210095)

Message sent from 8335 (LAMPORT_TIME=5, LOCAL_TIME=2024-02-14 16:53:39.210097)

Event happened in 8334 ! (LAMPORT_TIME=6, LOCAL_TIME=2024-02-14 16:53:39.210100)

Message received at 8336 (LAMPORT_TIME=6, LOCAL_TIME=2024-02-14 16:53:39.210174)

Message sent from 8336 (LAMPORT_TIME=7, LOCAL_TIME=2024-02-14 16:53:39.210266)

Message received at 8335 (LAMPORT_TIME=8, LOCAL_TIME=2024-02-14 16:53:39.210270)
```