|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

Experiment No. 04

|  |  |
| --- | --- |
| Semester | B.E. Semester VIII – Computer Engineering |
| Subject | Distributed Computing Lab |
| Subject Professor In-charge | Dr. Umesh Kulkarni |
| Assisting Professor | Prof. Prakash Parmar |
| Academic Year | 2024-25 |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |

**Title:** Simulate a distributed system with multiple nodes (processes or computers), each with its own local clock.

**Explanation:  
  
1. Introduction**

In a distributed system, multiple nodes (processes or computers) operate independently, each maintaining its own local clock. Due to hardware variations and environmental factors, these clocks tend to drift apart, leading to discrepancies in timekeeping. Clock synchronization is essential to ensure a consistent notion of time across all nodes, which is crucial for coordinating actions, maintaining event ordering, and ensuring system reliability.

**2. The Clock Synchronization Problem**

In a distributed system, the absence of a global clock leads to challenges in maintaining a consistent timeline. The two main issues are:

1. **Clock Drift** – The difference in time due to slight variations in hardware clocks.
2. **Clock Skew** – The difference between two clocks at a given instance.

To counteract these issues, synchronization mechanisms are employed to adjust the local clocks of all nodes and bring them into alignment.

**3. Clock Synchronization Techniques**

There are two primary methods of clock synchronization:

**A. External Synchronization**

* The nodes synchronize their clocks with a reference time source (e.g., UTC or GPS clocks).
* Example: Network Time Protocol (NTP).

**B. Internal Synchronization**

* The nodes synchronize with each other without an external reference.
* Example: Berkeley’s Algorithm, Cristian’s Algorithm.

Since our implementation focuses on **Berkeley’s Algorithm**, let’s understand it in detail.

**4. Berkeley’s Algorithm (Internal Synchronization)**

Berkeley’s Algorithm is a distributed clock synchronization algorithm designed to achieve internal synchronization in a network of nodes.

**Working Mechanism:**

1. **Master Node Selection:**
   * One node is designated as the master (coordinator). It is responsible for synchronizing the clocks of other nodes.
2. **Time Collection:**
   * The master node polls all other nodes to get their current local times.
   * Each node replies with its own clock time.
3. **Offset Calculation:**
   * The master calculates the average time of all nodes, ignoring outliers if necessary.
   * The offset for each node is determined as the difference between its local clock and the computed average time.
4. **Clock Adjustment:**
   * The master sends an adjustment value to all nodes, instructing them to modify their clocks accordingly.
   * Each node updates its clock by adding/subtracting the given offset.

**5. Implementation of Berkeley’s Algorithm in Java**

In this implementation, we simulate a distributed system with **multiple nodes** running on separate threads. The **master node** (node 0) gathers time information from all nodes, computes the **average time**, and adjusts the clocks of all nodes accordingly.

**6. Code Implementation Explanation**

The Java program consists of the following components:

1. **Node Class (Callable<Integer>)**
   * Each node has a **local clock** initialized with a random drift.
   * Implements call() method to return its clock time.
   * Provides a method to **adjust its time** when synchronization occurs.
2. **Clock Synchronization Process:**
   * The **master node (node 0)** collects times from all nodes.
   * Computes the **average time** from collected values.
   * Determines the required **offset** for each node.
   * Adjusts the clocks to achieve synchronization.

**CODE:  
  
import** **java.util.Random**;

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

**class** Node **implements** Callable<**Integer**> {

**private** **int** nodeId;

**private** **int** localTime;

**private** **Random** rand **=** **new** Random();

**public** Node(**int** nodeId) {

**this**.nodeId **=** nodeId;

**this**.localTime **=** 1000 **+** rand.nextInt(200); // Simulating different local times

    }

**public** **int** getTime() {

**return** localTime;

    }

**public** **void** adjustTime(**int** offset) {

        localTime **+=** offset;

    }

    @**Override**

**public** **Integer** call() {

**return** localTime;

    }

}

**public** **class** Main {

**public** **static** **void** main(**String**[] args) **throws** **InterruptedException**, **ExecutionException** {

**int** numNodes **=** 5;

**ExecutorService** executor **=** Executors.newFixedThreadPool(numNodes);

**Node**[] nodes **=** **new** **Node**[numNodes];

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

            nodes[i] **=** **new** Node(i);

        }

        // Master node is node[0]

**Node** masterNode **=** nodes[0];

        System.out.println("Master Node ID: 0, Time: " **+** masterNode.getTime());

        // Collect times from all nodes

        @**SuppressWarnings**("unchecked")

**Future**<**Integer**>[] futures **=** **new** **Future**[numNodes];

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

            futures[i] **=** executor.submit(nodes[i]);

        }

        // Calculate average time offset

**int** totalTime **=** 0;

**for** (**Future**<**Integer**> future **:** futures) {

            totalTime **+=** future.get();

        }

**int** avgTime **=** totalTime **/** numNodes;

        System.out.println("Calculated Average Time: " **+** avgTime);

        // Synchronize all nodes

**for** (**Node** node **:** nodes) {

**int** offset **=** avgTime **-** node.getTime();

            node.adjustTime(offset);

            System.out.println("Node " **+** node **+** " adjusted by " **+** offset **+** ", New Time: " **+** node.getTime());

        }

        executor.shutdown();

    }

}

**Output:**

