**Experiment-06**

**Aim: Develop a program for clock synchronization algorithms.**

**Tools:** Python, Google Colab

**Theory:** Berkeley's Algorithm is a distributed algorithm for computing the correct time in a network of computers. The algorithm is designed to work in a network where clocks may be running at slightly different rates, and some computers may experience intermittent communication failures.

The basic idea behind Berkeley's Algorithm is that each computer in the network periodically sends its local time to a designated "master" computer, which then computes the correct time for the network based on the received timestamps. The master computer then sends the correct time back to all the computers in the network, and each computer sets its clock to the received time.

There are several variations of Berkeley's Algorithm that have been proposed, but a common version of the algorithm is as follows −

- Each computer starts with its own local time, and periodically sends its time to the master computer.
- The master computer receives timestamps from all the computers in the network.
- The master computer computes the average time of all the timestamps it has received and sends the average time back to all the computers in the network.
- Each computer sets its clock to the time it receives from the master computer.
- The process is repeated periodically, so that over time, the clocks of all the computers in the network will converge to the correct time.

One benefit of Berkeley's Algorithm is that it is relatively simple to implement and understand. However, It has a limitation that the time computed by the algorithm is based on the network conditions and time of sending and receiving timestamps which can't be very accurate and also it has a requirement of a master computer which if failed can cause the algorithm to stop working.

There are other more advanced algorithms such as the Network Time Protocol (NTP) which use a more complex algorithm and also consider the network delay and clock drift to get a more accurate time.

**Result and Discussion:**

**Input:**

```
from datetime import datetime, timedelta

def berkeley_algorithm(nodes):
    # Calculate the average time (converted to timestamps)
    average_time = sum(node['time'].timestamp() for node in nodes) /
len(nodes)

    # Calculate the time difference for each node
    for node in nodes:
        node['offset'] = average_time - node['time'].timestamp()
        node['synchronized_time'] = node['time'] +
timedelta(seconds=node['offset'])

def synchronize_clocks(nodes):
    for node in nodes:
        # Synchronize the clock for each node
        node['synchronized_time'] = node['time'] +
timedelta(seconds=node['offset'])

def print_node_times(nodes):
    for node in nodes:
        print(f"Node {node['id']} - Local Time: {node['time']},
Synchronized Time: {node.get('synchronized_time', 'Not synchronized')}")

if __name__ == "__main__":
    # Example with three nodes
    nodes = [
        {'id': 1, 'time': datetime.now()},
        {'id': 2, 'time': datetime.now() + timedelta(seconds=5)},
        {'id': 3, 'time': datetime.now() - timedelta(seconds=3)}
    ]

    print("Original Node Times:")
    print_node_times(nodes)

    berkeley_algorithm(nodes)
    synchronize_clocks(nodes)

    print("\nAfter Berkeley Algorithm:")
    print_node_times(nodes)
```

**Output:**

```
Original Node Times:
Node 1 - Local Time: 2024-02-26 09:34:37.229477, Synchronized Time: Not synchronized
Node 2 - Local Time: 2024-02-26 09:34:42.229481, Synchronized Time: Not synchronized
Node 3 - Local Time: 2024-02-26 09:34:34.229487, Synchronized Time: Not synchronized

After Berkeley Algorithm:
Node 1 - Local Time: 2024-02-26 09:34:37.229477, Synchronized Time: 2024-02-26 09:34:37.896148
Node 2 - Local Time: 2024-02-26 09:34:42.229481, Synchronized Time: 2024-02-26 09:34:37.896148
Node 3 - Local Time: 2024-02-26 09:34:34.229487, Synchronized Time: 2024-02-26 09:34:37.896148
```

**Learning Outcomes:** The student should have the ability to

LO1: Describe Berkeley's algorithm for clock synchronization
LO2: Write a program for clock synchronization algorithms.

**Course Outcomes:** Upon completion of the course students will be able to describe and implement clock synchronization algorithms

**Conclusion:** The basic implementation of the Berkeley algorithm for clock synchronization in a distributed system. It calculates the average time, determines time differences for each node, and synchronizes their clocks accordingly. The example involves three nodes with different local times, and the program prints the original and synchronized times after applying the Berkeley algorithm.

**For Faculty Use**

| Correction Parameters | Formative Assessment [40%] | Timely completion of Practical [ 40%] | Attendance / Learning Attitude [20%] | |
|---|---|---|---|---|
| Marks Obtained | | | | |