20CYS402 – Distributed Systems and Cloud Computing

Lab 2

Student Name: R Subramanian

Register Number: CH.EN.U4CYS22043

Objective

To simulate and understand clock synchronization in distributed systems by implementing:

1. **Berkeley Algorithm** – for *physical clock synchronization*, ensuring consistent time across distributed nodes.
2. **Lamport's Algorithm** – for *logical clock synchronization*, ensuring correct event ordering among processes.

2.1. Physical Clock Synchronization – Berkeley Algorithm

Aim

To simulate a time daemon polling multiple computer nodes to get their local times, compute the average, and send correction values so that all nodes synchronize to the same time.

Algorithm Steps

1. The time daemon requests the current local time from all nodes.
2. It collects all the times, including its own.
3. It calculates the **average time** of all nodes.
4. The daemon sends **correction values** (difference from the average) to each node.
5. Each node adjusts its clock accordingly.
6. Display the synchronized time and correction values for each node.

CODE

Berkeley Algorithm.py > ...

```python
from datetime import datetime, timedelta

def time_str_to_minutes(t):
    h, m = map(int, t.split(":"))
    return h * 60 + m

def minutes_to_time_str(m):
    h = m // 60
    m = m % 60
    return f"{h:02d}:{m:02d}"

def berkeley_algorithm(daemon_time, node_times):
    all_times = [time_str_to_minutes(daemon_time)] + [time_str_to_minutes(t) for t in node_times]
    avg_time = sum(all_times) // len(all_times)
    corrections = [avg_time - t for t in all_times]

    print("\nAfter Synchronization...\n")
    print(f"Time Daemon : {minutes_to_time_str(avg_time)}")
    for i, corr in enumerate(corrections[1:], start=1):
        print(f"Node {i}: {minutes_to_time_str(avg_time)} [Correction Value: {corr:+} mins]")

daemon = "15:00"
nodes = ["15:10", "14:50", "15:25"]

print(f"Time Daemon : {daemon}")
for i, t in enumerate(nodes, start=1):
    print(f"Node {i}: {t}")

berkeley_algorithm(daemon, nodes)
```

Output

```
PS C:\Users\amma\Documents\DSCC\Lab2> python -u "c:\Users\amma\Documents\DSCC\Lab2\Berkeley Algorithm.py"
Time Daemon : 15:00
Node 1: 15:10
Node 2: 15:50
Node 3: 15:25

After Synchronization...

Time Daemon : 15:21
Node 1: 15:21 [Correction Value: +11 mins]
Node 2: 15:21 [Correction Value: -29 mins]
Node 3: 15:21 [Correction Value: -4 mins]
```

2.2. Logical Clock Synchronization – Lamport's Algorithm

Aim

To simulate logical clock synchronization between multiple processes to maintain consistent event ordering in a distributed system.

Algorithm Steps

1. Each process maintains a **logical clock** (counter).
2. For every internal event, increment the clock by 1.
3. When sending a message, attach the current clock value.
4. On receiving a message, update the receiver's clock to:
   `max(receiver_clock, sender_clock) + 1`

5. Display the logical clock updates for each event to ensure proper ordering.

CODE

```python
Lamport Algorithm.py > ...
class Process:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, pid):
        self.pid = pid
        self.clock = 0
        self.events = []

    Tabnine | Edit | Test | Explain | Document
    def internal_event(self):
        self.clock += 1
        self.events.append((self.clock, "internal"))

    Tabnine | Edit | Test | Explain | Document
    def send_event(self, receiver):
        self.clock += 1
        message = (self.clock, self.pid)
        receiver.receive_event(message)
        self.events.append((self.clock, f"send to P{receiver.pid}"))

    Tabnine | Edit | Test | Explain | Document
    def receive_event(self, message):
        recv_clock, sender_id = message
        self.clock = max(self.clock, recv_clock) + 1
        self.events.append((self.clock, f"recv from P{sender_id}"))

    Tabnine | Edit | Test | Explain | Document
    def print_events(self):
        print(f"Process {self.pid} Event History:")
        for clock, event in self.events:
            print(f"  Clock={clock} -> {event}")
        print()

P1 = Process(1)
P2 = Process(2)

P1.internal_event()
P1.send_event(P2)
P2.internal_event()
P2.send_event(P1)
P1.internal_event()

P1.print_events()
P2.print_events()

```

Output

```
PS C:\Users\amma\Documents\DSCC\Lab2> python -u "c:\Users\amma\Documents\DSCC\Lab2\Lamport Algorithm.py"
Process 1 Event History:
  Clock=1 -> internal
  Clock=2 -> send to P2
  Clock=6 -> recv from P2
  Clock=7 -> internal

Process 2 Event History:
  Clock=3 -> recv from P1
  Clock=4 -> internal
  Clock=5 -> send to P1
```

Conclusion

The lab successfully demonstrated two fundamental synchronization techniques in distributed systems:

- **Berkeley Algorithm:** Achieved physical clock synchronization by averaging the clock times and applying appropriate corrections.
- **Lamport's Algorithm:** Maintained logical clock order to ensure consistent event sequencing across processes.

These algorithms are essential for **coordination, consistency, and reliability** in distributed environments.