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. So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks. The physical clocks are used to adjust the time of nodes.Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.
The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

- External clock synchronization is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
- Internal clock synchronization is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.

- Centralized is the one in which a time server is used as a reference. The single time server propagates its time to the nodes and all the nodes adjust the time accordingly. It is dependent on a single time server so if that node fails, the whole system will lose synchronization. Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.
- Distributed is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

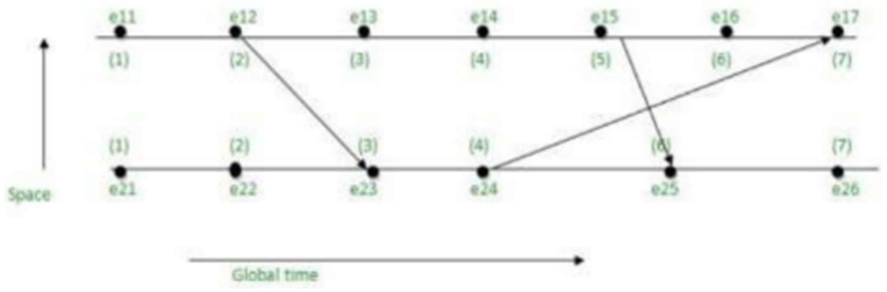
Lamport‘s Logical Clock
Lamport‘s Logical Clock was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

<u>Algorithm:</u>
● Happened before relation(->): a -> b, means ‗a‘ happened before ‗b‘.
● Logical Clock: The criteria for the logical clocks are: ○ [C1]: Ci (a) < Ci(b), [Ci -> Logical Clock, If ‗a‘ happened before ‗b‘, then time of ‗a‘ will be less than ‗b‘ in a particular process.] ○ [C2]: Ci(a) <Cj(b), [Clock value of Ci(a) is less than Cj(b)]

Reference:
● Process: Pi
● Event: Eij, where i is the process in number and j: jth event in the ith process.
● tm: vector time span for message m.
● Ci vector clock associated with process Pi, the jth element is Ci[j] and contains Pi‗s latest value for the current time in process Pj.
● d: drift time, generally d is 1.

Implementation Rules[IR]:
● [IR1]: If a -> b [‗a' happened before ‗b' within the same process] then, Ci(b) =Ci(a) + d
● [IR2]: Cj = max(Cj, tm + d) [If there's more number of processes, then tm = value of Ci(a), Cj = max value between Cj and tm + d]

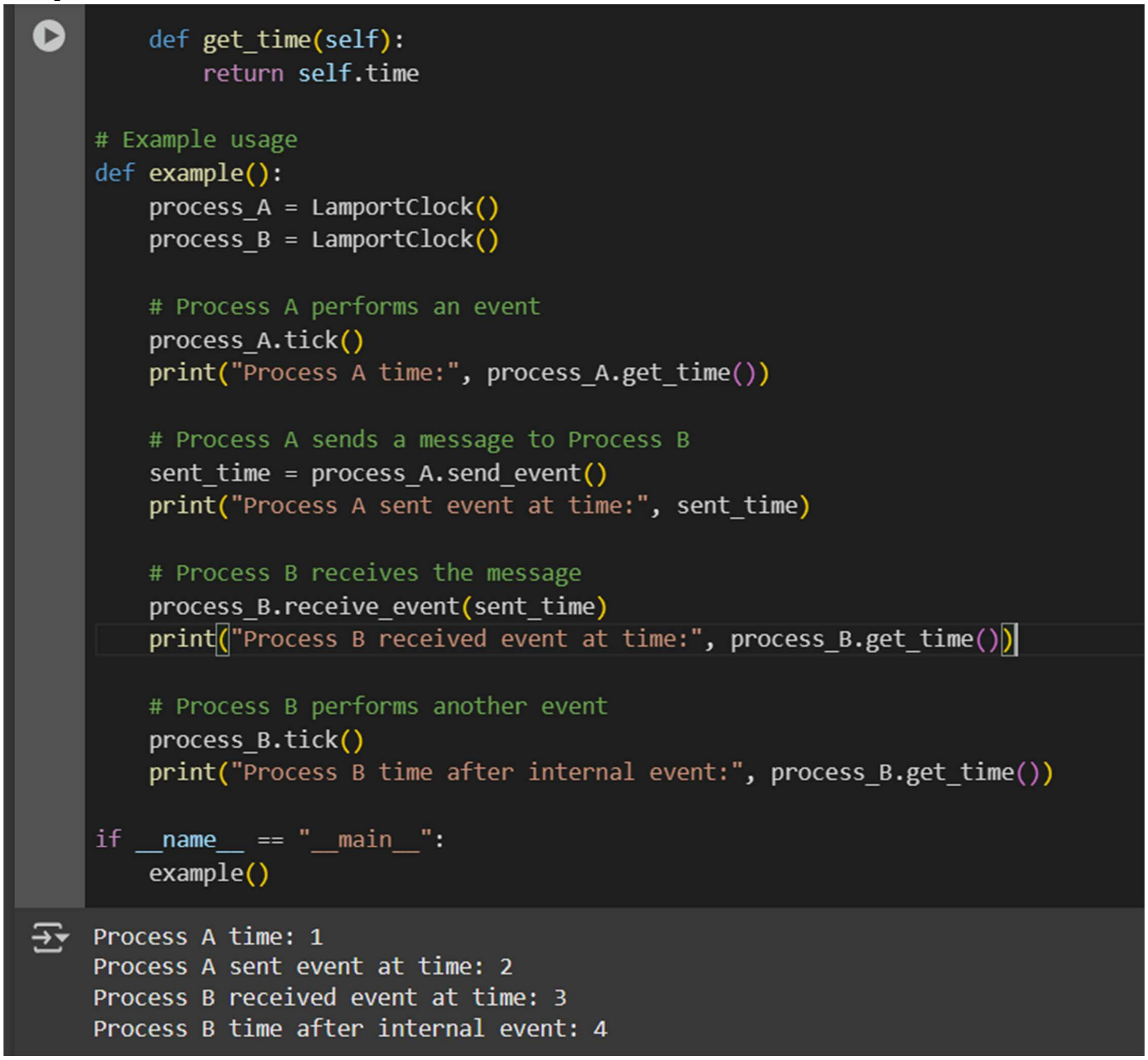
For Example,



Take the starting value as 1, since it is the 1st event and there is no incomingvalue at the starting point:
○ e11 = 1
○ e21 = 1
● The value of the next point will go on increasing by d (d = 1), if there is no incoming value i.e., to follow [IR1].
○ e12 = e11 + d = 1 + 1 = 2
○ e13 = e12 + d = 2 + 1 = 3
○ e14 = e13 + d = 3 + 1 = 4
○ e15 = e14 + d = 4 + 1 = 5
○ e16 = e15 + d = 5 + 1 = 6
○ e22 = e21 + d = 1 + 1 = 2
○ e24 = e23 + d = 3 + 1 = 4
○ e26 = e25 + d = 6 + 1 = 7
● When there will be an incoming value, then follow [IR2] i.e., take the maximum value between Cj and Tm + d.
○ e17 = max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, maximum among 7 and 5 is 7]
○ e23 = max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, maximum among 3 and 3 is 3]
○ e25 = max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, maximum among 5 and 6 is 6]

Program:
```
class LamportClock:
    def __init__(self):
        self.time = 0

    def tick(self):
        """Increment the clock for an internal event."""
        self.time += 1

    def send_event(self):
        """Increment the clock and return the timestamp for a message being sent."""
        self.time += 1
        return self.time

    def receive_event(self, received_time):
        """Update the clock when receiving a message."""
        self.time = max(self.time, received_time) + 1

    def get_time(self):
        return self.time

# Example usage
def example():
    process_A = LamportClock()
    process_B = LamportClock()

    # Process A performs an event
    process_A.tick()
    print("Process A time:", process_A.get_time())

    # Process A sends a message to Process B
    sent_time = process_A.send_event()
    print("Process A sent event at time:", sent_time)

    # Process B receives the message
    process_B.receive_event(sent_time)
    print("Process B received event at time:", process_B.get_time())

    # Process B performs another event
    process_B.tick()
    print("Process B time after internal event:", process_B.get_time())

if __name__ == "__main__":
    example()
```

Output:



```
    def get_time(self):
        return self.time

# Example usage
def example():
    process_A = LamportClock()
    process_B = LamportClock()

    # Process A performs an event
    process_A.tick()
    print("Process A time:", process_A.get_time())

    # Process A sends a message to Process B
    sent_time = process_A.send_event()
    print("Process A sent event at time:", sent_time)

    # Process B receives the message
    process_B.receive_event(sent_time)
    print("Process B received event at time:", process_B.get_time())

    # Process B performs another event
    process_B.tick()
    print("Process B time after internal event:", process_B.get_time())

if __name__ == "__main__":
    example()
```

```
Process A time: 1
Process A sent event at time: 2
Process B received event at time: 3
Process B time after internal event: 4
```

Conclusion:
Thus, we had Successfully Implemented Load Balancing Algorithm.