Distributed Systems Assignment

Submitted By

Joypal Taid

Roll No: 210710007021
Branch: Computer Science and Engineering
Batch: 2021-2025

Question: Implement Lamport's Logical Clock using Python Programming Language

Explanation of the Algorithm:

The algorithm is pretty straightforward and works as such:

1. Each process in the system maintains its own logical clock, which is essentially a counter (initially set to zero) that is incremented for each event it experiences.
2. When a process does work, it increments its own clock value by a certain unit (usually 1).
3. When a process sends a message, it includes its current clock value with the message.
4. When a process receives a message, it updates its clock to be the maximum of its own clock and the received clock value from the message, and then increments it by 1. This ensures that the receiving process logically happens after the sending process and any other events that the sender knew about.

Implementation in Python:

Code:

```
class  LamportsClock:
        def  __init__(self, process_id):
        self.process_id = process_id
        self.clock = 0

        def  tick(self):
        #Increment the clock on internal events.
        self.clock += 1
        print(f"Process {self.process_id} tick: {self.clock}")

        def send_event(self):
        #Simulate sending a message with the current clock value.
        self.clock += 1
        print(f"Process {self.process_id} sends event with clock {self.clock}")
        return self.clock

        def  receive_event(self, received_clock):
        """
        Simulate receiving a message and updating the clock.
        The clock is updated to the maximum of the current clock or received clock + 1.
        """
        self.clock = max(self.clock, received_clock) + 1
        print(f"Process {self.process_id} receives event and updates clock to {self.clock}")


if __name__ == "__main__":
        # Create two processes
        process_A = LamportsClock(process_id="A")
        process_B = LamportsClock(process_id="B")

        # Simulate events in Process A
        process_A.tick()  # Internal event in A
        sent_clock = process_A.send_event()  # A sends a message

        # Simulate events in Process B
        process_B.tick()  # Internal event in B
        process_B.receive_event(sent_clock)  # B receives the message from A

        # Another event in Process A
        process_A.tick()
```

```
# B sends a message to A
sent_clock = process_B.send_event()
process_A.receive_event(sent_clock)  # A receives the message from B
```

Output:

Process A tick: 1
Process A sends event with clock 2
Process B tick: 1
Process B receives event and updates clock to 3
Process A tick: 3
Process B sends event with clock 4
Process A receives event and updates clock to 5