# Practical-3

**Aim:**

The primary goal of this assignment is to simulate a distributed system where multiple processes communicate through message passing.

The focus is on implementing Lamport's Logical Clock algorithm to order events across processes, thereby ensuring consistent event ordering in a distributed environment lacking a global clock.

**Theory:**

**Distributed Systems and Event Ordering:**

In distributed systems, processes operate independently and communicate by exchanging messages. Since these processes do not share a global clock, determining the order of events across different processes is challenging.

Without a proper mechanism, it becomes difficult to resolve conflicts, manage dependencies, or synchronize actions across the system.

**Lamport's Logical Clock:**

Leslie Lamport introduced the concept of Logical Clocks to address the problem of ordering events in distributed systems. Each process maintains its own logical clock, which is a simple integer value. The clock is incremented according to specific rules when certain events occur:

1. **Internal Event**: When a process performs an internal event, it increments its logical clock by 1.
2. **Send Event**: When a process sends a message, it increments its logical clock by 1 and attaches this updated clock value to the message.

3. **Receive Event**: Upon receiving a message, a process sets its logical clock to the maximum of its own clock and the received clock, then increments the result by 1.

**Vector Clocks:**

While Lamport's Logical Clocks can order events, they may not always capture causal relationships between events. Vector Clocks, an extension of Lamport's clocks, are used to address this limitation. In a Vector Clock system:

- Each process maintains an array (vector) where each element represents the clock value of a particular process.
- During communication, the vector clocks are exchanged, and the receiving process updates its vector by taking the element-wise maximum of its own vector and the received vector.

This method ensures that all causal relationships between events are captured, making Vector Clocks more accurate than Lamport's Logical Clocks in complex scenarios.

**Functions & Modules Used:**

1. **Message Class:**
    - **Attributes**: senderId, receiverId, timeStamp, vectorClockOfSender.
    - **Purpose**: Represents a message sent between processes. Includes the logical clock value and vector clock of the sender.
    - **Methods**:
        - getSenderId(): Returns the sender's ID.
        - getReceiverId(): Returns the receiver's ID.
        - getTimeStamp(): Returns the timestamp of the message.
        - getVectorClockOfSender(): Returns the vector clock of the sender.
        - toString(): Returns a string representation of the message.

2. **SharedBuffer Class:**

    - **Attributes**: ownerId, messageQueue.
    - **Purpose**: Acts as a mailbox for a process, storing incoming messages in a queue.
    - **Methods**:
        - addMessage(Message message): Adds a message to the queue.
        - retrieveMessage(): Retrieves and removes a message from the queue or returns null if the queue is empty.
        - logEvent(String event): Logs events to a file named LamportLog.txt.

3. **ProcessThread Class:**

    - **Attributes**: ownerId, sharedBuffer, logicalClock, vectorClock, rand, iterations.
    - **Purpose**: Represents a process in the distributed system. It performs internal, send, and receive events, updating its logical and vector clocks accordingly.
    - **Methods**:
        - run(): Executes the process's operations for a specified number of iterations.
        - performInternalEvent(): Handles internal events by incrementing the logical and vector clocks.
        - performSendEvent(): Handles send events by sending messages to other processes and updating the clocks.
        - performReceiveEvent(): Handles receive events by updating the clocks based on the received message.
        - logEvent(String event): Logs the process's events to the LamportLog.txt file.
        - getClockValue(): Returns the current logical clock value.
        - getVectorClock(): Returns the current vector clock.

4. **LamportLogicalClock Class:**

    - **Attributes**: buffers.
    - **Purpose**: Serves as the main class, responsible for initializing and managing processes in the simulation.
    - **Methods**:
        - main(String[] args): Initializes processes, assigns them to threads, and starts the simulation.
        - getBuffer(int processId): Retrieves the buffer (mailbox) of a specified process.

**Analysis:**

**Simulation Setup:**

- **Number of Processes**: Three processes are used in the simulation, each running in its own thread.
- **Iterations**: Each process performs a fixed number of iterations (10 in this case).
- **Events**: The simulation randomly selects one of three possible events (internal, send, or receive) during each iteration.

**Execution Flow:**

1. **Initialization**: Each process starts with a logical clock value of 0 and an initialized vector clock (array) where all elements are set to 0.
2. **Event Execution**:
    - **Internal Event**: The process increments its logical clock and updates its vector clock for its own ID.
    - **Send Event**: The process increments its logical and vector clocks, creates a message containing these clocks, and sends it to another process's buffer.
    - **Receive Event**: The process retrieves a message from its buffer, updates its clocks by taking the maximum values, and logs the event.
3. **Logging**: Each event is logged to a LamportLog.txt file, capturing the event type, current logical clock, and vector clock values.

**Output Stored In Log File:**

Simulation started at: 2024-08-31 23:07:48

Process 2 sent Message from Process 2 to Process 0 at Time 1, Vector Clock: [0, 0, 1]

Process 0 received Message from Process 2 to Process 0 at Time 1, Vector Clock: [0, 0, 1]. Updated clock: 2, Vector Clock: [1, 0, 1]

Process 1's mailbox is empty or no message received.

Process 1 sent Message from Process 1 to Process 2 at Time 1, Vector Clock: [0, 1, 0]

Process 1's mailbox is empty or no message received.

Process 0's mailbox is empty or no message received.

Process 2 received Message from Process 1 to Process 2 at Time 1, Vector Clock: [0, 1, 0]. Updated clock: 2, Vector Clock: [0, 1, 2]

Process 1 performed internal event at time 2, Vector Clock: [0, 2, 0]

Process 0's mailbox is empty or no message received.

Process 2's mailbox is empty or no message received.

Process 0 sent Message from Process 0 to Process 2 at Time 3, Vector Clock: [2, 0, 1]

Process 2 sent Message from Process 2 to Process 1 at Time 3, Vector Clock: [0, 1, 3]

Process 1 received Message from Process 2 to Process 1 at Time 3, Vector Clock: [0, 1, 3]. Updated clock: 4, Vector Clock: [0, 3, 3]

Process 0 sent Message from Process 0 to Process 1 at Time 4, Vector Clock: [3, 0, 1]

Process 1 sent Message from Process 1 to Process 0 at Time 5, Vector Clock: [0, 4, 3]

Process 2 performed internal event at time 4, Vector Clock: [0, 1, 4]

Process 2 performed internal event at time 5, Vector Clock: [0, 1, 5]

Process 0 received Message from Process 1 to Process 0 at Time 5, Vector Clock: [0, 4, 3]. Updated clock: 6, Vector Clock: [4, 4, 3]

Process 1 sent Message from Process 1 to Process 2 at Time 6, Vector Clock: [0, 5, 3]

Process 0's mailbox is empty or no message received.

Process 2 performed internal event at time 6, Vector Clock: [0, 1, 6]

Process 0 performed internal event at time 7, Vector Clock: [5, 4, 3]

Process 1 sent Message from Process 1 to Process 0 at Time 7, Vector Clock: [0, 6, 3]

Process 0 received Message from Process 1 to Process 0 at Time 7, Vector Clock: [0, 6, 3]. Updated clock: 8, Vector Clock: [6, 6, 3]

Process 1 received Message from Process 0 to Process 1 at Time 4, Vector Clock: [3, 0, 1]. Updated clock: 8, Vector Clock: [3, 7, 3]

Process 2 sent Message from Process 2 to Process 0 at Time 7, Vector Clock: [0, 1, 7]

Process 2 received Message from Process 0 to Process 2 at Time 3, Vector Clock: [2, 0, 1]. Updated clock: 8, Vector Clock: [2, 1, 8]

Process 1 performed internal event at time 9, Vector Clock: [3, 8, 3]

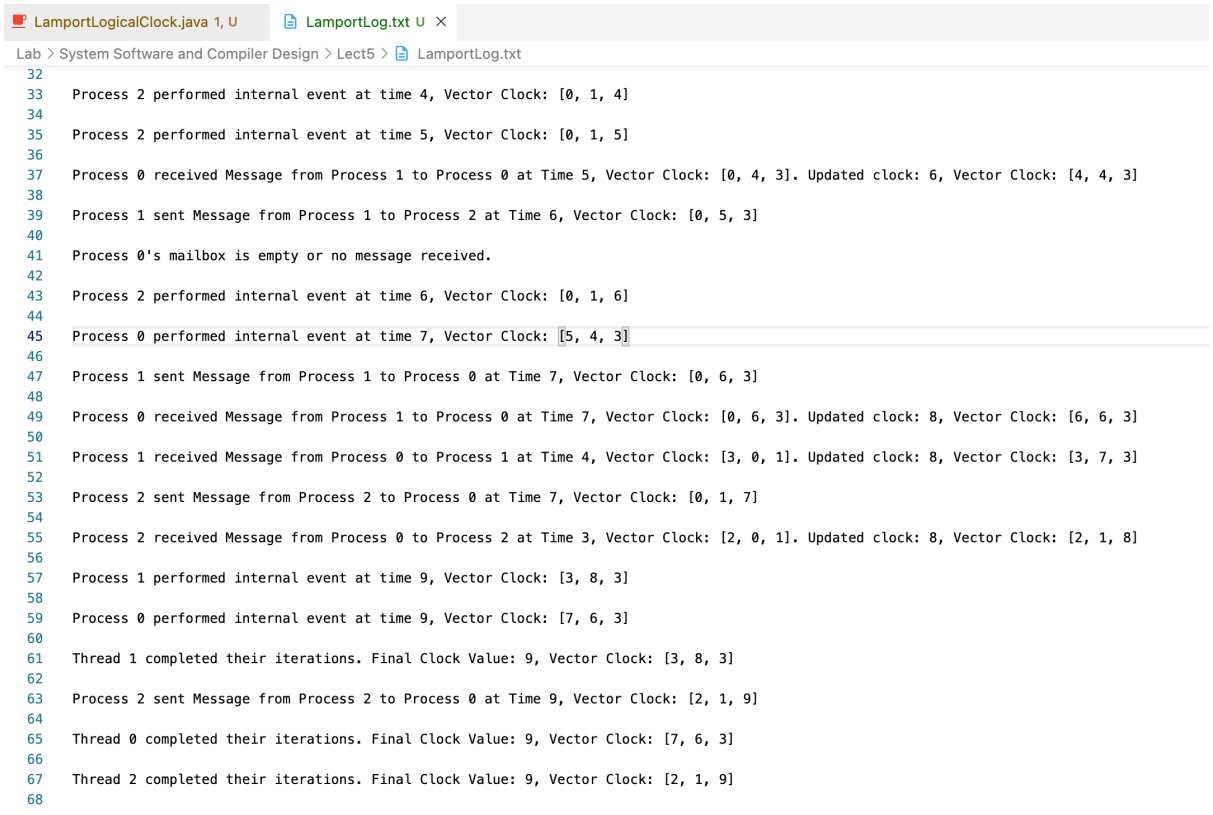
Process 0 performed internal event at time 9, Vector Clock: [7, 6, 3]

Thread 1 completed their iterations. Final Clock Value: 9, Vector Clock: [3, 8, 3]

Process 2 sent Message from Process 2 to Process 0 at Time 9, Vector Clock: [2, 1, 9]

Thread 0 completed their iterations. Final Clock Value: 9, Vector Clock: [7, 6, 3]

Thread 2 completed their iterations. Final Clock Value: 9, Vector Clock: [2, 1, 9]




**Calculations:**

Initial state:
VC: P0 [0,0,0], P1 [0,0,0], P2 [0,0,0]
LC: P0 0, P1 0, P2 0

1. Process 2 sends message to Process 0 Calculation: max(0, 0) + 1 = 1
   VC: P0 [0,0,0], P1 [0,0,0], P2 [0,0,1]
   LC: P0 0, P1 0, P2 1

2. Process 0 receives message from Process 2 Calculation: max(0, 1) + 1 = 2
   VC: P0 [1,0,1], P1 [0,0,0], P2 [0,0,1]
   LC: P0 2, P1 0, P2 1
3. Process 1 sends message to Process 2 Calculation: max(0, 0) + 1 = 1
   VC: P0 [1,0,1], P1 [0,1,0], P2 [0,0,1]
   LC: P0 2, P1 1, P2 1
4. Process 2 receives message from Process 1 Calculation: max(1, 1) + 1 = 2
   VC: P0 [1,0,1], P1 [0,1,0], P2 [0,1,2]
   LC: P0 2, P1 1, P2 2
5. Process 1 internal event Calculation: max(1, 1) + 1 = 2
   VC: P0 [1,0,1], P1 [0,2,0], P2 [0,1,2]
   LC: P0 2, P1 2, P2 2
6. Process 0 sends message to Process 2 Calculation: max(2, 2) + 1 = 3
   VC: P0 [2,0,1], P1 [0,2,0], P2 [0,1,2]
   LC: P0 3, P1 2, P2 2
7. Process 2 sends message to Process 1 Calculation: max(2, 2) + 1 = 3
   VC: P0 [2,0,1], P1 [0,2,0], P2 [0,1,3]
   LC: P0 3, P1 2, P2 3
8. Process 1 receives message from Process 2 Calculation: max(2, 3) + 1 = 4
   VC: P0 [2,0,1], P1 [0,3,3], P2 [0,1,3]
   LC: P0 3, P1 4, P2 3
9. Process 0 sends message to Process 1 Calculation: max(3, 3) + 1 = 4
   VC: P0 [3,0,1], P1 [0,3,3], P2 [0,1,3]
   LC: P0 4, P1 4, P2 3
10. Process 1 sends message to Process 0 Calculation: max(4, 4) + 1 = 5
    VC: P0 [3,0,1], P1 [0,4,3], P2 [0,1,3]
    LC: P0 4, P1 5, P2 3
11. Process 2 internal event Calculation: max(3, 3) + 1 = 4
    VC: P0 [3,0,1], P1 [0,4,3], P2 [0,1,4]
    LC: P0 4, P1 5, P2 4
12. Process 2 internal event Calculation: max(4, 4) + 1 = 5
    VC: P0 [3,0,1], P1 [0,4,3], P2 [0,1,5]
    LC: P0 4, P1 5, P2 5

13. Process 0 receives message from Process 1 Calculation: max(4, 5) + 1 = 6
    VC: P0 [4,4,3], P1 [0,4,3], P2 [0,1,5]
    LC: P0 6, P1 5, P2 5
14. Process 1 sends message to Process 2 Calculation: max(5, 5) + 1 = 6
    VC: P0 [4,4,3], P1 [0,5,3], P2 [0,1,5]
    LC: P0 6, P1 6, P2 5
15. Process 2 internal event Calculation: max(5, 5) + 1 = 6
    VC: P0 [4,4,3], P1 [0,5,3], P2 [0,1,6]
    LC: P0 6, P1 6, P2 6
16. Process 0 internal event Calculation: max(6, 6) + 1 = 7
    VC: P0 [5,4,3], P1 [0,5,3], P2 [0,1,6]
    LC: P0 7, P1 6, P2 6
17. Process 1 sends message to Process 0 Calculation: max(6, 6) + 1 = 7
    VC: P0 [5,4,3], P1 [0,6,3], P2 [0,1,6]
    LC: P0 7, P1 7, P2 6
18. Process 0 receives message from Process 1 Calculation: max(7, 7) + 1 = 8
    VC: P0 [6,6,3], P1 [0,6,3], P2 [0,1,6]
    LC: P0 8, P1 7, P2 6
19. Process 1 receives message from Process 0 Calculation: max(7, 4) + 1 = 8
    VC: P0 [6,6,3], P1 [3,7,3], P2 [0,1,6]
    LC: P0 8, P1 8, P2 6
20. Process 2 sends message to Process 0 Calculation: max(6, 6) + 1 = 7
    VC: P0 [6,6,3], P1 [3,7,3], P2 [0,1,7]
    LC: P0 8, P1 8, P2 7
21. Process 2 receives message from Process 0 Calculation: max(7, 3) + 1 = 8
    VC: P0 [6,6,3], P1 [3,7,3], P2 [2,1,8]
    LC: P0 8, P1 8, P2 8
22. Process 1 internal event Calculation: max(8, 8) + 1 = 9
    VC: P0 [6,6,3], P1 [3,8,3], P2 [2,1,8]
    LC: P0 8, P1 9, P2 8
23. Process 0 internal event Calculation: max(8, 8) + 1 = 9
    VC: P0 [7,6,3], P1 [3,8,3], P2 [2,1,8]
    LC: P0 9, P1 9, P2 8

24. Process 2 sends message to Process 0 Calculation: max(8, 8) + 1 = 9
    VC: P0 [7,6,3], P1 [3,8,3], P2 [2,1,9]
    LC: P0 9, P1 9, P2 9

Final state:
VC: P0 [7,6,3], P1 [3,8,3], P2 [2,1,9]
LC: P0 9, P1 9, P2 9

**Analysis of Results:**

- The simulation ensures that all events are correctly ordered according to the logical clocks.
- When processes communicate, the receiving process updates its logical clock based on the message's timestamp, ensuring consistency across the system.
- Vector Clocks provide additional information, helping identify the causality of events, which is particularly useful in complex scenarios.

**Conclusion:**

The implementation of Lamport's Logical Clock in a simulated distributed system successfully demonstrates how logical clocks can be used to order events in environments without a global clock.

The extension to Vector Clocks provides a more nuanced view of event causality, enabling better conflict resolution and event ordering.

Through this assignment, the foundational principles of distributed systems, message passing, and clock synchronization have been explored, highlighting the importance of logical clocks in achieving consistent event ordering.

The code's modular structure, including the Message, SharedBuffer, ProcessThread, and LamportLogicalClock classes, ensures clarity and reusability, adhering to best practices in software development.

The results, as logged in LamportLog.txt, confirm the correct operation of the logical and vector clocks, meeting the assignment's objectives and learning outcomes.

Further extensions, such as real-time communication or network-based implementations, could build on this foundation, adding real-world relevance and complexity.