Experiment 04

- **Aim:** To implement Lamport's Timestamp algorithm in student portal using fileupload

- **Objectives:**

1) To implement clock synchronization between multiple processes
2) To implement Lamport's clock synchronization algorithm
3) To get a total ordering of event based on logical timestamps

- **Theory:**

Events: In a distributed system, there are three primary types of events:

Local Event: These are events that occur within a single process, such as executing a command.

Send Event: When one process sends a message to another process.

Receive Event: When a process receives a message from another process.
Defining the "Happened Before" Relationship:

The "->" symbol is used to represent the "happened before" relationship. If event A occurs causally before event B, then timestamp(A) < timestamp(B).
Causality in this context refers to the causal relationship or the chain of events that connect one occurrence to another in a way that implies causation.
The algorithm provides a partial order of events because it's often impossible to definitively establish the causality between all events.
Basic Rules for "Happened Before" Relationship:

For local events: $a \rightarrow b$ if time(a) < time(b). This means that if the timestamp of event A is less than the timestamp of event B, A causally precedes B.

For send and receive events:

If process P1 sends a message to process P2, then send(message) → receive(message). This relationship signifies that a send event must happen before a receive event for the same message.
Transitive property: If a → b and b → c, then a → c. In other words, if A causally precedes B and B causally precedes C, then A causally precedes C.

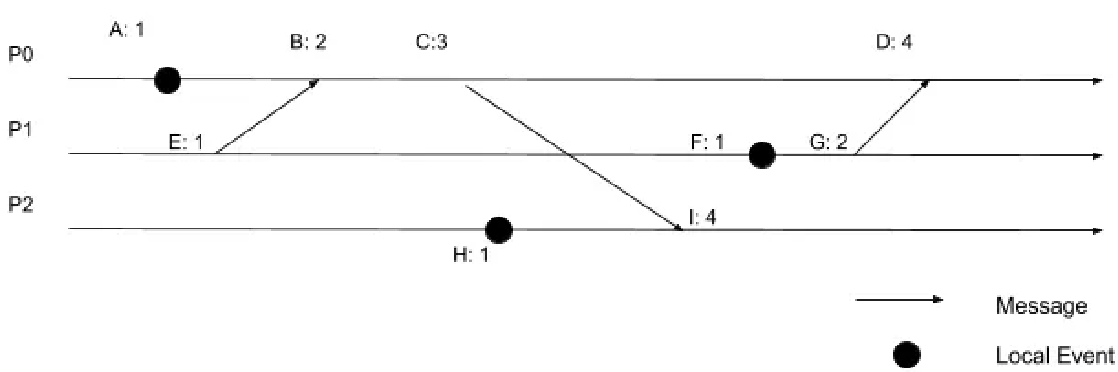
Updating Logical Timestamps:

Logical timestamps are updated according to the following rules:

Before executing an event, the process increments the logical timestamp by 1: Logical_timestamp = Logical_timestamp + 1. This ensures that events are assigned increasing timestamps.

During a send event, the process increments the logical timestamp by 1 and sends the timestamp along with the message: Logical_timestamp = Logical_timestamp + 1.

During a receive event, the recipient updates its logical timestamp to the maximum value between its own Logical_timestamp and the timestamp received with the message, and then increments the timestamp by 1: Logical_timestamp = max(Logical_timestamp, time_received) + 1



The image illustrates a communication network, showcasing the intricate exchange of messages between distinct nodes. It comprises three horizontal lines denoted as P0, P1, and P2, each hosting eight labeled points: A1, B2, C3, D4, E1, F1, G2, and H1. These points are interconnected by lines, bearing numerical labels that likely signify the message flow. Notably, the diagram employs black circles to mark local events and arrows to signify message transmissions. Such visual representations are commonly employed in computer science to provide a clear visualization of data propagation within a network.

Here, we could say that E ->B as timestamp(E) < timestamp(B), and there is a causal path from A to B. Similarly, C->I, F->G and so on. But if we consider the events, A and E we have a timestamp(A) = timestamp(E). But there is no causal path from A to E or E to A. Similarly, for the events, A and G, we have timestamp(A) < timestamp(G). But there is no causal path from A to G or G to A. These events are called concurrent events and a pair of concurrent events doesn’t have a causal path from one event to another.

❖ Algorithm

1) Initialize a local logical clock to 0 for each process in the distributed system.
2) When an parallel event occurs:
   a) clock = clock + 1
3) When a sequential event occurs
   a) Pass timestamp of the calling server in request (received_ts)
   b) clock = max(received_ts, clock) + 1

- **Code Snippet:**

1) Reusable lamport class

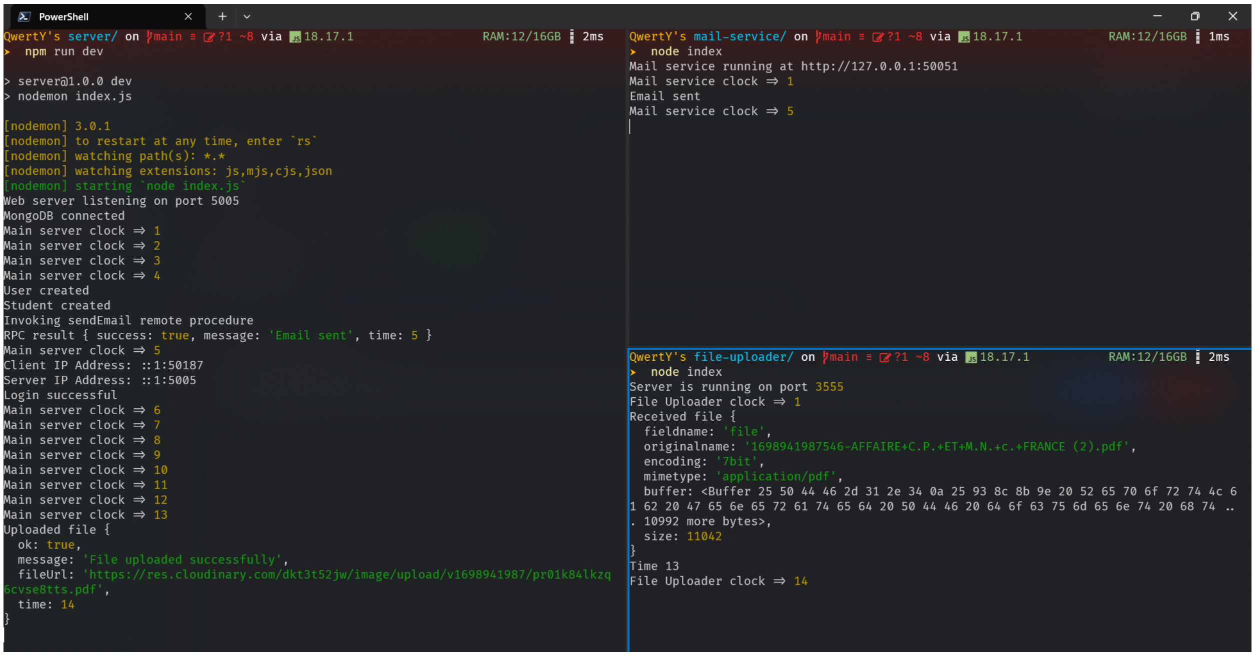
server > lamport.js > LamportClock > tick

```javascript
class LamportClock {
    constructor() {
        this.clock = 0;
    }

    tick() {
        this.clock++;
        this.printTime();
    }

    getTime() {
        return this.clock;
    }

    updateTime(receivedTime) {
        const behind = this.clock < receivedTime;
        this.clock = Math.max(this.clock, receivedTime) + 1;
        this.printTime();
        return behind;
    }

    printTime() {
        console.log("Main server clock =>", this.clock);
    }
}

export default LamportClock;
```

2) Updating timestamp

```
app.post("/upload", multerUploader.single("file"), async (req, res) => {
    try {
        if (!req.file) {
            return res.status(400).json({ message: "No file uploaded" });
        }

        console.log("Received file", req.file);

        cloudinary.uploader
            .upload_stream((err, result) => {
                const fileUrl = result?.secure_url;

                const { time } = req.body;
                console.log("Time", time);

                if (!time) {
                    // parallel request
                    lamportClock.tick();
                } else {
                    // sequential request
                    lamportClock.updateTime(time);
                }

                return res.json({
                    ok: true,
                    message: "File uploaded successfully",
                    fileUrl,
                    time: lamportClock.getTime(),
                });
            })
            .end(req.file.buffer);
    } catch (e) {
        res.status(500).json({ message: "Internal server error" });
    }
});
```

❖ Output



❖ Conclusion:

Thereby we have successfully implemented lamport in our student portal system for uploading files such that there is synchronization between time at which files were submitted by the student and the time they were received by the faculty.

❖ References:

[1] https://en.wikipedia.org/wiki/Lamport_timestamp
[2] https://lamport.azurewebsites.net/pubs/time-clocks.pdf
[3] https://www.geeksforgeeks.org/lamports-logical-clock/
[4] https://medium.com/big-data-processing/lamport-timestamps-833a077e1a86
[5] Lamport Clock