

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 " \rightarrow " symbol is used to represent the "happened before" relationship. If event A occurs causally before event B, then $\text{timestamp}(A) < \text{timestamp}(B)$.

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 $\text{time}(a) < \text{time}(b)$. 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 $\text{send}(\text{message}) \rightarrow \text{receive}(\text{message})$. This relationship signifies that a send event must happen before a receive event for the same message.

Transitive property: If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. In other words, if A causally precedes B and B causally precedes C, then A causally precedes C.

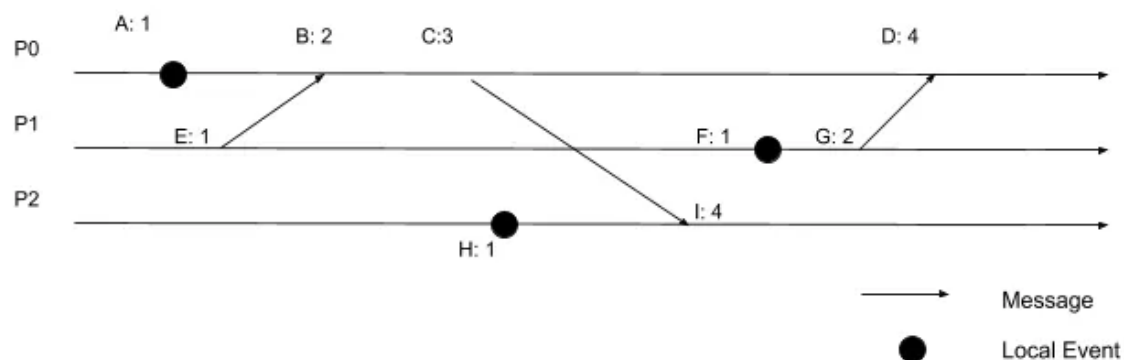
Updating Logical Timestamps:

Logical timestamps are updated according to the following rules:

Before executing an event, the process increments the logical timestamp by 1:
 $\text{Logical_timestamp} = \text{Logical_timestamp} + 1$. 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: $\text{Logical_timestamp} = \text{Logical_timestamp} + 1$.

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: $\text{Logical_timestamp} = \max(\text{Logical_timestamp}, \text{time_received}) + 1$



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 $E \rightarrow B$ as $\text{timestamp}(E) < \text{timestamp}(B)$, and there is a causal path from A to B. Similarly, $C \rightarrow I$, $F \rightarrow G$ and so on. But if we consider the events, A and E we have $\text{timestamp}(A) = \text{timestamp}(E)$. But there is no causal path from A to E or E to A. Similarly, for the events, A and G, we have $\text{timestamp}(A) < \text{timestamp}(G)$. 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) $\text{clock} = \text{clock} + 1$
- 3) When a sequential event occurs
 - a) Pass timestamp of the calling server in request (`received_ts`)
 - b) $\text{clock} = \max(\text{received_ts}, \text{clock}) + 1$

❖ Code Snippet:

1) Reusable lamport class

```
server >  lamport.js >  LamportClock >  tick
7   class LamportClock {
6     constructor() {
5       this.clock = 0;
4     }
3
2     tick() {
1       this.clock++;
8     this.printTime();
1    }
2
3    getTime() {
4      return this.clock;
5    }
6
7    updateTime(receivedTime) {
8      const behind = this.clock < receivedTime;
9      this.clock = Math.max(this.clock, receivedTime) + 1;
10     this.printTime();
11     return behind;
12   }
13
14   printTime() {
15     console.log("Main server clock =>", this.clock);
16   }
17 }
18
19 export default LamportClock;
20
```

2) Updating timestamp

```
19 app.post("/upload", multerUploader.single("file"), async (req, res) => {
18   try {
17     if (!req.file) {
16       return res.status(400).json({ message: "No file uploaded" });
15     }
14
13     console.log("Received file", req.file);
12
11     cloudinary.uploader
10       .upload_stream((err, result) => {
9         const fileUrl = result?.secure_url;
8
7         const { time } = req.body;
6         console.log("Time", time);
5
4         if (!time) {
3           // parallel request
2           lamportClock.tick();
1         } else {
66           // sequential request
1         lamportClock.updateTime(time);
2
3         }
4
5         return res.json({
6           ok: true,
7           message: "File uploaded successfully",
8           fileUrl,
9           time: lamportClock.getTime(),
10         });
11       })
12       .end(req.file.buffer);
13   } catch (e) {
14     res.status(500).json({ message: "Internal server error" });
15   }
16 });
```

❖ Output

```
PowerShell
Querty's server/ on pmain = 18.17.1 RAM:12/16GB 2ms
> npm run dev
> server@1.0.0 dev
> nodemon index.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
Web server listening on port 5005
MongoDB connected
Main server clock => 1
Main server clock => 2
Main server clock => 3
Main server clock => 4
User created
Student created
Invoking sendEmail remote procedure
RPC result { success: true, message: 'Email sent', time: 5 }
Main server clock => 5
Client IP Address: ::1:50187
Server IP Address: ::1:5005
Login successful
Main server clock => 6
Main server clock => 7
Main server clock => 8
Main server clock => 9
Main server clock => 10
Main server clock => 11
Main server clock => 12
Main server clock => 13
Uploaded file {
  ok: true,
  message: 'file uploaded successfully',
  fileId: 'https://res.cloudinary.com/dkt3t52jw/image/upload/v1698941987/pr0lk84lkzq/1698941987.pdf',
  time: 14
}

Querty's mail-service/ on pmain = 18.17.1 RAM:12/16GB 1ms
> node index
Mail service running at http://127.0.0.1:50051
Mail service clock => 1
Email sent
Mail service clock => 5

Querty's file-uploader/ on pmain = 18.17.1 RAM:12/16GB 2ms
> node index
Server is running on port 3555
File Uploader clock => 1
Received file {
  filename: 'file',
  originalname: '1698941987546-AFFAIRE+C.P.+ET+M.N.+c.+FRANCE (2).pdf',
  encoding: '7bit',
  mimetype: 'application/pdf',
  buffer: <Buffer 25 50 44 46 2d 31 2e 34 0a 25 93 8c 8b 9e 20 52 65 70 6f 72 74 4c 6
1 62 20 47 65 6e 65 72 61 74 65 64 20 50 44 46 20 64 6f 63 75 6d 65 6e 74 20 68 74 ..
. 10992 more bytes>,
  size: 11042
}
Time 13
File Uploader clock => 14
```

❖ 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](#)