Tushar Deshmukh
Final Year CSE PE27
1032201698

**TITLE:**
Write a C program to implement Lamport's Logical Clock Algorithm.

**AIM:**
To study and implement Lamport's Logical Clock Algorithm.

**OBJECTIVES:**
1. To understand the working of Lamport's Logical Clock Algorithm.

**THEORY:**
Logical and Physical Clocks:
Logical Clock: A logical clock is a mechanism used to order events in a distributed system. Unlike physical clocks that represent actual time, logical clocks do not rely on real-time measurements. Instead, they assign virtual timestamps to events based on their ordering. Logical clocks are typically implemented using counters that are incremented for each event, ensuring that causally related events receive sequential timestamps.

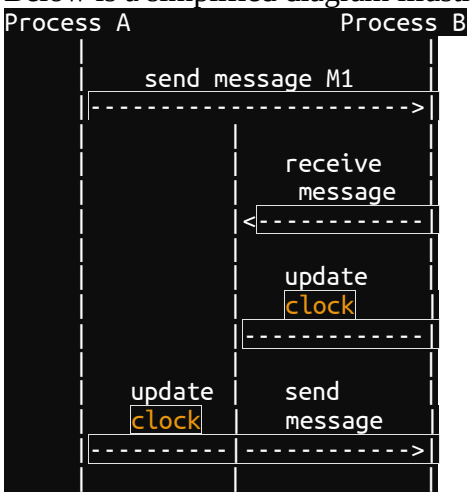
Physical Clock: A physical clock, on the other hand, represents real time and is based on physical hardware, such as quartz crystals or atomic clocks. These clocks provide timestamps that correspond to the actual passage of time, enabling accurate timekeeping in computing systems.

Lamport's Clock Correction:
Lamport's Clock: Lamport's logical clocks are a method for ordering events in a distributed system. Each process in the system maintains a logical clock value, which is incremented whenever an event occurs. When a process sends a message, it includes its current logical clock value. Upon receiving a message, the recipient updates its own logical clock to reflect the maximum of its current value and the value received in the message, plus one.

Clock Correction: Lamport's clock correction is a technique used to adjust the logical clocks of processes in a distributed system to ensure consistent ordering of events. When processes communicate, they exchange their logical clock values. If a process receives a message with a logical clock value greater than its own, it adjusts its logical clock to be greater than or equal to the received value plus one.

Tushar Deshmukh
Final Year CSE PE27
1032201698

Below is a simplified diagram illustrating Lamport's clock correction:




Consider two processes, A and B, in a distributed system. Process A sends a message to process B. Before sending the message, process A increments its logical clock value to 5. Upon receiving the message, process B updates its logical clock value to max(own value, received value) + 1 = max(3, 5) + 1 = 6.

This ensures that events are ordered correctly across processes, even in a distributed environment where each process has its own notion of time.

**INPUT:**
processes and their logical clocks with need of correction

**OUTPUT**:
the corrected logical clock output by Lamport's algorithm

**PLATFORM:**

Windows:

VSCode
Windows Subsystem for Linux (Ubuntu 22.04 LTS)

**PROGRAMMING LANGUAGE:**
C++ Language

**CODE:**

```cpp
#include <iostream>
#include <list>

#define max_msgs 10

using namespace std;
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

```
class Message
{
public:
    int sid;
    int rid;
    int se;
    int re;
    int C;
    bool sent;
    bool received;

    Message(int se, int re, int sid, int rid)
    {
        this->se = se;
        this->re = re;
        this->sid = sid;
        this->rid = rid;
        this->C = 0;
        this->sent = false;
        this->received = false;
    }

    void displayMessage()
    {
        cout << "Sender- p" << this->sid << ", e" << this->se << "," <<
this->C << ", ; Reciever- p " << this->rid << ", e" << this->re << "; sent
" << this->sent << ", received " << this->received << endl;
    }
};

class Process
{
private:
    static int nextId;

public:
    int id;
    int d;
    int localC;
    int event;

    Process(int d = 1)
    {
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

```
        this->id = nextId++;
        this->d = d;
        this->localC = 0;
        this->event = 0;
    }

    void displayProcess()
    {
        cout << "Process " << this->id << ", d " << this->d << ", localC "
<< this->localC << ", event " << this->event << endl;
    }
};

int Process::nextId = 1;

int main()
{
    list<Process> processes;
    list<Message> messages;

    int np = 0;
    cout << "Enter number of processes: ";
    cin >> np;

    for (int i = 0; i < np; ++i)
    {
        int d = 0;
        cout << "Enter d for process " << (i + 1) << ": ";
        cin >> d;
        processes.push_back(Process(d));
    }

    int nm = 0;
    cout << "Enter number of messages: ";
    cin >> nm;

    for (int i = 0; i < nm; ++i)
    {
        int sid = 0, rid = 0;
        int se = 0, re = 0;
        cout << "Enter sender id for message " << (i + 1) << ": ";
        cin >> sid;
        cout << "Event number: ";
        cin >> se;
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

```
        cout << "Enter reciever id for message " << (i + 1) << ": ";
        cin >> rid;
        cout << "Event number: ";
        cin >> re;
        messages.push_back(Message(se, re, sid, rid));
    }

    for (auto &p : processes)
    {
        p.displayProcess();
    }

    for (auto &m : messages)
    {
        m.displayMessage();
    }

    cout << "Event\t";
    for (int i = 0; i < np; i++)
    {
        cout << "p" << i + 1 << "\t";
    }
    cout << endl;

    for (int i = 0; i <= max_msgs; i++)
    {
        cout << "e" << i << "\t";
        for (auto &p : processes)
        {
            cout << p.localC << "\t";
            p.event += 1;
            p.localC += p.d;

            // send
            for (auto &m : messages)
            {
                if (m.sid == p.id && m.se == p.event)
                {
                    m.C = p.localC; // piggyback sender time
                    m.sent = true;
                }
            }

            // rec
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

```
            for (auto &m : messages)
            {
                if (m.rid == p.id && m.re == p.event)
                {
                    // set receiver time
                    p.localC = max(p.localC, m.C);
                    p.localC += p.d;
                    m.received = true;
                    // m.displayMessage();
                }
            }
        }
        cout << endl;
    }

    // for (auto &p : processes)
    // {
    //     p.displayProcess();
    // }

    for (auto &m : messages)
    {
        m.displayMessage();
    }

    return 0;
}
```

**INPUT**: Three processes with d = {6,8,10}. Four messages m = {m1: p1 1 → p2 2, m2: p2 3 → p3 4, m3: p3 6 → p2 7, m4: p2 8 → p1 9}. Run clock.

Tushar Deshmukh
Final Year CSE PE27
1032201698

**OUTPUT:**

```
cs572go@LAPTOP-J3V82B9N: /mnt/d/DC/LCA3
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA3$ g++ lamport.cpp -o lamport
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA3$ ./lamport
Enter number of processes: 3
Enter d for process 1: 6
Enter d for process 2: 8
Enter d for process 3: 10
Enter number of messages: 4
Enter sender id for message 1: 1
Event number: 1
Enter reciever id for message 1: 2
Event number: 2
Enter sender id for message 2: 2
Event number: 3
Enter reciever id for message 2: 3
Event number: 4
Enter sender id for message 3: 3
Event number: 6
Enter reciever id for message 3: 2
Event number: 7
Enter sender id for message 4: 2
Event number: 8
Enter reciever id for message 4: 1
Event number: 9
Process 1, d 6, localC 0, event 0
Process 2, d 8, localC 0, event 0
Process 3, d 10, localC 0, event 0
Sender- p1, e1,0, ; Reciever- p 2, e2; sent 0, received 0
Sender- p2, e3,0, ; Reciever- p 3, e4; sent 0, received 0
Sender- p3, e6,0, ; Reciever- p 2, e7; sent 0, received 0
Sender- p2, e8,0, ; Reciever- p 1, e9; sent 0, received 0
Event   p1      p2      p3
e0      0       0       0
e1      6       8       10
e2      12      24      20
e3      18      32      30
e4      24      40      50
e5      30      48      60
e6      36      56      70
e7      42      78      80
e8      48      86      90
e9      92      94      100
e10     98      102     110
Sender- p1, e1,6, ; Reciever- p 2, e2; sent 1, received 1
Sender- p2, e3,32, ; Reciever- p 3, e4; sent 1, received 1
Sender- p3, e6,70, ; Reciever- p 2, e7; sent 1, received 1
Sender- p2, e8,86, ; Reciever- p 1, e9; sent 1, received 1
cs572go@LAPTOP-J3V82B9N:/mnt/d/DC/LCA3$
```

Tushar Deshmukh
Final Year CSE PE27
1032201698

**CONCLUSION**:
Thus, Lamport's algorithm is successfully implemented.

**FAQs**
1. What is clock skew and clock drift?
Answer:
Clock Skew: Clock skew refers to the inconsistency or difference in time measurements between two clocks that should ideally be synchronized. In distributed systems, clock skew can occur due to various factors such as network latency, hardware limitations, or software inaccuracies. It can lead to incorrect timestamping of events and affect the overall system behavior.

Clock Drift: Clock drift is the gradual deviation of a clock's timekeeping accuracy from an ideal or reference clock over time. This deviation can be caused by factors such as temperature changes, aging of components, or manufacturing variations. Clock drift can result in clocks running faster or slower than expected, leading to inaccuracies in timekeeping.

2. How to convert a logical clock into physical clock?
Answer:
Converting a logical clock into a physical clock involves mapping the logical timestamps generated by the logical clock to actual time values provided by a physical clock. This process requires synchronization between the logical and physical clocks to ensure consistency and accuracy in timestamping events. One common approach is to periodically synchronize the logical clock with the physical clock by adjusting the logical clock's offset and drift rate based on the observed differences between the two clocks over time.

3. What is happened before event in Lamport's clock?
Answer:
In Lamport's logical clocks, the "happened-before" relationship defines the ordering of events in a distributed system. An event A is said to have happened before event B if one of the following conditions holds:

- Event A occurs before event B in the same process.
- Event A is the sending of a message by one process, and event B is the receipt of that message by another process.
- There exists an event C such that event A happened before event C and event C happened before event B.

In essence, the happened-before relationship captures the causal relationship between events in a distributed system, allowing Lamport's clocks to order events consistently across different processes, even in the absence of global time synchronization.