```
/*
Name: Rohit Saini
Erp: 1032200897
Panel: C
RollNo: PC-41 */
```

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 : what is a logical and physical lock.what is lamports clock correction,explain with diagram and example.

A logical clock and a physical clock, as they relate to Lamport's clock synchronization, are terms used to differentiate between two types of clocks that are used for ordering events in distributed systems. Let me explain both concepts, including Lamport's clock synchronization, with a diagram and an example:

1. Physical Clock:

A physical clock, also known as a real-time clock, is a clock that tracks actual time in the system. This can be a system clock, a wall clock, or any other hardware or software clock that provides the current time.

2. Logical Clock:

A logical clock is an abstract clock that does not necessarily correspond to real-time. Instead, it's used to order events based on their logical order or causality. Logical clocks are typically used in distributed systems to establish a partial ordering of events, even if they occur on different physical clocks. Lamport's logical clock is one example of a logical clock.

Lamport's Clock Synchronization:

Lamport's clock synchronization algorithm is a method used to assign logical timestamps to events in a distributed system. It ensures that events happening at different nodes of the system can be correctly ordered, even when there is no global time or global clock.

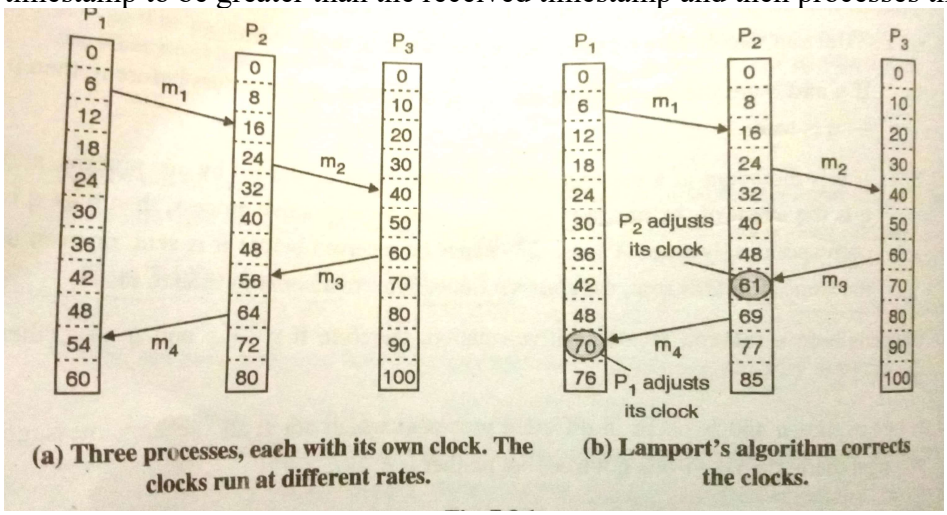
Here's how Lamport's clock synchronization works:

Event Timestamps: Each event in the system is timestamped with a logical timestamp. These timestamps are integers.

Event Ordering: Events are ordered based on their logical timestamps. If event A has a lower timestamp than event B, it is considered to have occurred before event B.

Message Exchange: When a process sends a message to another process, it includes its own timestamp with the message.

Timestamp Update: Upon receiving a message, the receiving process updates its own timestamp to be greater than the received timestamp and then processes the message.



(a) Three processes, each with its own clock. The clocks run at different rates.

(b) Lamport's algorithm corrects the clocks.

For the above:
When m1: No time synchronization needed
When m2: No time synchronization needed
When m3:

Process 2 has synchronized its clock
Process 1 : 0 6 12 18 24 30 36 42 48 54 60
Process 2 : 0 8 16 24 32 40 48 61 69 77 85
Process 3 : 0 10 20 30 40 50 60 70 80 90 100

When m4:

Process 1 has synchronized its clock
Process 1 : 0 6 12 18 24 30 36 42 48 65 71
Process 2 : 0 8 16 24 32 40 48 61 69 77 85
Process 3 : 0 10 20 30 40 50 60 70 80 90 100

INPUT : processes and their logical clocks with need of correction

CODE:

```cpp
#include <iostream>
#include <vector>
using namespace std;

class Process
{
private:
    vector<int> logicalClock;
    int id, clockRate;

public:
    int time;

    Process(int id, int time, int clockRate)
    {
        this->id = id;
        this->time = time;
        this->clockRate = clockRate;
        for (int i = 0; i <= time; i++)
        {
            logicalClock.push_back(i * clockRate);
        }
    }

    void print()
    {
        cout << "Process " << this->id << " :  ";
        for (int i = 0; i <= this->time; i++)
        {
            cout << logicalClock[i] << "  ";
        }
    }

    int getTime(int timeStamp) { return this->logicalClock[timeStamp];
}
```

```
    void updateLogicalClock(int senderTime, int receiverTime)
    {
        if (receiverTime > senderTime)
        {
            cout << "No time synchronization needed" << endl;
            return;
        }
        int timeStamp = this->searchTime(receiverTime);
        int start = senderTime + 1;
        for (int i = timeStamp; i <= time; i++)
        {
            logicalClock[i] = start + ((i - timeStamp) * clockRate);
        }
        cout << "Process " << this->id << " has synchronized its clock"
<< endl;
    }

    int searchTime(int clockTime)
    {
        for (int i = 0; i <= this->time; i++)
        {
            if (logicalClock[i] == clockTime)
                return i;
        }
        return -1;
    }
};

void printAllProcesses(vector<Process *> processes)
{
    for (Process *p : processes)
    {
        p->print();
        cout << endl;
    }
}
```

```cpp
int main()
{
    int n, time = 10;
    cout << "Enter number of processes : ";
    cin >> n;
    vector<Process *> processes;
    for (int i = 0; i < n; i++)
    {
        int clockRate;
        cout << "Enter clock rate of Process " << i + 1 << " : ";
        cin >> clockRate;
        processes.push_back(new Process(i + 1, time, clockRate));
    }
    printAllProcesses(processes);

    // Let's see messages now
    char choice = 'Y';
    while (choice == 'Y')
    {
        int senderTime, receiverTime, sender, receiver, timeStamp;
        cout << "Enter sender id : ";
        cin >> sender;
        cout << "Enter sender time : ";
        cin >> senderTime;
        cout << "Enter receiver id : ";
        cin >> receiver;
        cout << "Enter receiver time : ";
        cin >> receiverTime;
        processes[receiver - 1]->updateLogicalClock(senderTime,
receiverTime);
        printAllProcesses(processes);
        cout << "Do you want to send more messages (Y/N): ";
        cin >> choice;
    }
    return 0;
```

```
}
```

OUTPUT : the corrected logical clock output by lamports algorithm

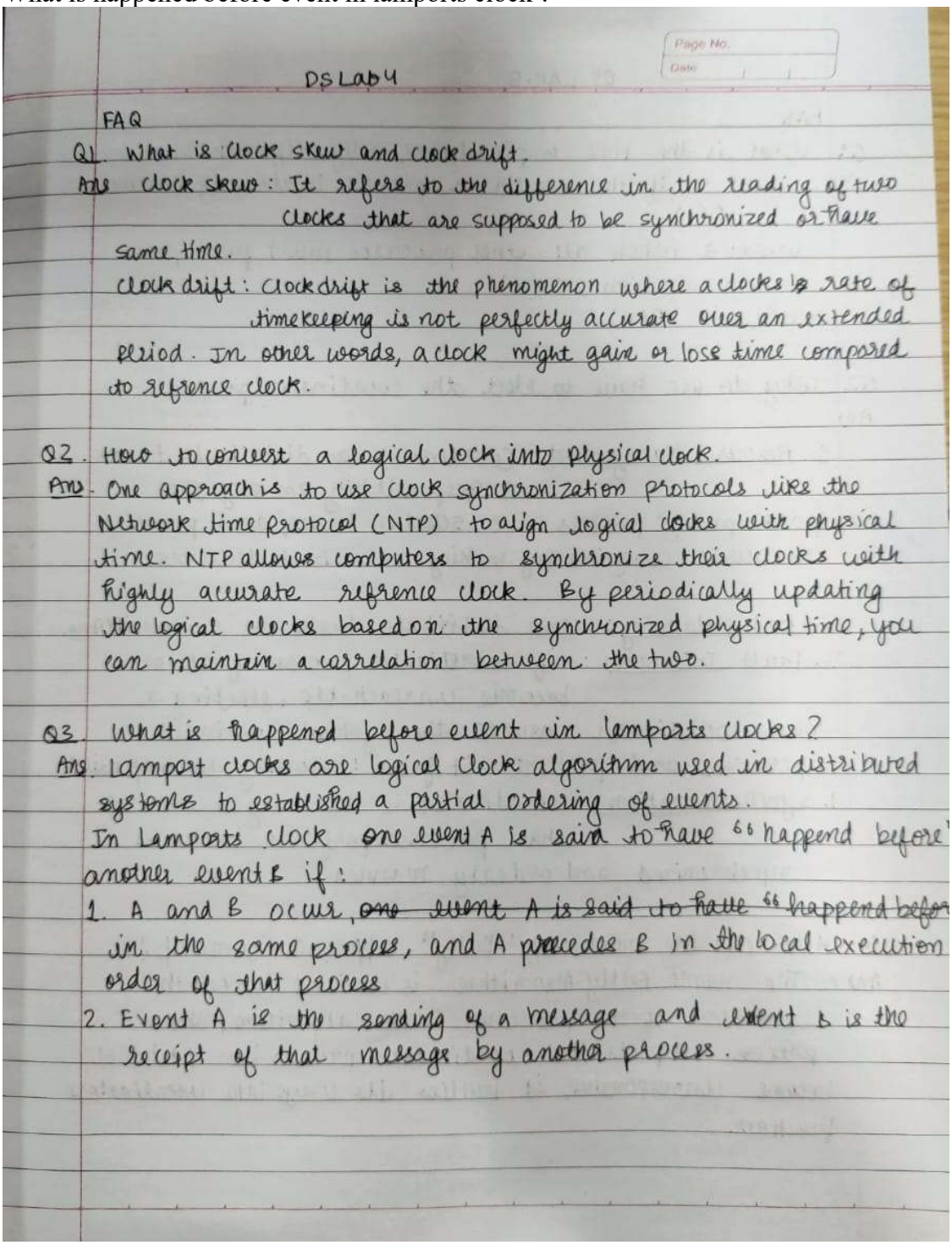
```
rohit@DESKTOP-3DK43OM MINGW64 ~/Documents/GitHub/sem_7/dc/lab4 (main)
$ ./lamport_clock.exe
Enter number of processes : 3
Enter clock rate of Process 1 : 6
Enter clock rate of Process 2 : 8
Enter clock rate of Process 3 : 10
Process 1 :  0  6  12  18  24  30  36  42  48  54  60
Process 2 :  0  8  16  24  32  40  48  56  64  72  80
Process 3 :  0  10  20  30  40  50  60  70  80  90  100
Enter sender id : 1
Enter sender time : 6
Enter receiver id : 2
Enter receiver time : 16
No time synchronization needed
Process 1 :  0  6  12  18  24  30  36  42  48  54  60
Process 2 :  0  8  16  24  32  40  48  56  64  72  80
Process 3 :  0  10  20  30  40  50  60  70  80  90  100
Do you want to send more messages (Y/N): Y
Enter sender id : 2
Enter sender time : 24
Enter receiver id : 3
Enter receiver time : 40
No time synchronization needed
Process 1 :  0  6  12  18  24  30  36  42  48  54  60
Process 2 :  0  8  16  24  32  40  48  56  64  72  80
Process 3 :  0  10  20  30  40  50  60  70  80  90  100
Do you want to send more messages (Y/N): Y
Enter sender id : 3
Enter sender time : 60
Enter receiver id : 2
Enter receiver time : 56
Process 2 has synchronized its clock
Process 1 :  0  6  12  18  24  30  36  42  48  54  60
Process 2 :  0  8  16  24  32  40  48  61  69  77  85
Process 3 :  0  10  20  30  40  50  60  70  80  90  100
Do you want to send more messages (Y/N): Y
Enter sender id : 2
Enter sender time : 64
Enter receiver id : 1
Enter receiver time : 54
Process 1 has synchronized its clock
Process 1 :  0  6  12  18  24  30  36  42  48  65  71
Process 2 :  0  8  16  24  32  40  48  61  69  77  85
Process 3 :  0  10  20  30  40  50  60  70  80  90  100
Do you want to send more messages (Y/N): N
```

PLATFORM : UNIX,

PROGRAMMING LANGUAGE : C Language
CONCLUSION : Thus, lamports algorithm is successfully implemented.

FAQs

1. What is clock skew and clock drift
2. How to convert a logical clock into physical clock
3. What is happened before event in lamports clock ?



DS Lab 4

FAQ

Q1. What is clock skew and clock drift.

Ans Clock skew : It refers to the difference in the reading of two clocks that are supposed to be synchronized or have same time.

Clock drift : Clock drift is the phenomenon where a clocks's rate of timekeeping is not perfectly accurate over an extended period. In other words, a clock might gain or lose time compared to refrence clock.

Q2. How to convert a logical clock into physical clock.

Ans. One approach is to use clock synchronization protocols like the Network time protocol (NTP) to align logical clocks with physical time. NTP allows computers to synchronize their clocks with highly accurate refrence clock. By periodically updating the logical clocks based on the synchronized physical time, you can maintain a correlation between the two.

Q3. What is happened before event in lamports clocks ?

Ans. Lamport clocks are logical clock algorithm used in distributed systems to established a partial ordering of events.

In Lamports clock one event A is said to have "happend before" another event B if :

1. A and B occur, ~~one event A is said to have "happend before~~ in the same process, and A precedes B in the local execution order of that process
2. Event A is the sending of a message and event B is the receipt of that message by another process.