| **KARAN SUNIL SHAH** | **2020400048** |
| --- | --- |
| **KAIVAL KETAN SHAH** | **2020400047** |
| **MIRAT SHAH** | **2020400049** |

Experiment No. 3:

**Lamport Logical Clock**

# Aim

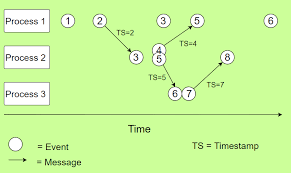
To implement a lamport logical clock in a program.

# Introduction

Lamport’s Logical Clock is a procedure to determine the order of events occurring in a distributed computer system.

Leslie Lamport invented a simple mechanism by which the happened-before ordering can be captured numerically. A Lamport logical clock is a numerical software counter value maintained in each process.

Conceptually, this logical clock can be thought of as a clock that only has meaning in relation to messages moving between processes. When a process receives a message, it re-synchronizes its logical clock with that sender.



# Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main()

{

int arr1[11];

int arr2[11];

int arr3[11];

int temp, size1, size2, size3;

printf("Enter the Starting Time of First Process: ");

scanf("%d", &temp);

printf("Enter the Step Size of First Process : ");

scanf("%d", &size1);

arr1[0] = temp;

for (int i = 1; i < 11; i++)

{

temp = temp + size1;

arr1[i] = temp;

}

printf("Enter the Starting Time of Second Process : ");

scanf("%d", &temp);

printf("Enter the Step Size of Second Process : ");

scanf("%d", &size2);

arr2[0] = temp;

for (int i = 1; i < 11; i++)

{

temp = temp + size2;

arr2[i] = temp;

}

printf("Enter the Starting Time of Third Process : ");

scanf("%d", &temp);

printf("Enter the Step Size of Third Process : ");

scanf("%d", &size3);

arr3[0] = temp;

for (int i = 1; i < 11; i++)

{

temp = temp + size3;

arr3[i] = temp;

}

printf("\nP1\tP2\tP3\n");

for (int i = 0; i < 11; i++)

{

printf("%d\t%d\t%d\n", arr1[i], arr2[i], arr3[i]);

}

printf("\n");

int arr[2][4];

printf("Enter the Index of path : \n");

for (int i = 0; i < 4; i++)

{

scanf("%d", &arr[0][i]);

scanf("%d", &arr[1][i]);

}

int flag = 1;

for (int i = 0; i < 4; i++)

{

if (flag == 1)

{

int t1 = arr1[arr[0][i]];

int t2 = arr2[arr[1][i]];

if (t1 < t2)

{

printf("%d -> %d is Valid path between TimeLine 1 to TimeLine 2\n", t1, t2);

}

else

{

printf("%d -> %d is Invalid path between TimeLine 1 to TimeLine 2\n", t1, t2);

}

}

else if (flag == 2)

{

int t1 = arr2[arr[0][i]];

int t2 = arr3[arr[1][i]];

if (t1 < t2)

{

printf("%d -> %d is Valid path between TimeLine 2 to TimeLine 3\n", t1, t2);

}

else

{

printf("%d -> %d is Invalid path between TimeLine 2 to TimeLine 3\n", t1, t2);

}

}

else if (flag == 3)

{

int t1 = arr3[arr[0][i]];

int t2 = arr2[arr[1][i]];

if (t1 < t2)

{

printf("%d -> %d is Valid path between TimeLine 3 to TimeLine 2\n", t1, t2);

}

else

{

printf("%d -> %d is Invalid path between TimeLine 3 to TimeLine 2\n", t1, t2);

arr2[arr[1][i]] = arr3[arr[0][i]] + 1;

printf("arr2[arr[1][%d]] -> %d\n", i, arr2[arr[1][i]]);

printf("arr[1][%d] -> %d\n", i, arr[1][i]);

for (int j = arr[1][i] + 1; j < 11; j++)

{

arr2[j] = arr2[j - 1] + size2;

}

}

}

else if (flag == 4)

{

int t1 = arr2[arr[0][i]];

int t2 = arr1[arr[1][i]];

if (t1 < t2)

{

printf("%d -> %d is Valid path between TimeLine 2 to TimeLine 1 \n", t1, t2);

}

else

{

printf("%d -> %d is Invalid path between TimeLine 2 to TimeLine 1\n", t1, t2);

arr1[(arr[0][i]) + 1] = arr2[(arr[1][i]) - 1] + 1;

printf("arr1[arr[0][%d]] -> %d\n", i, arr1[arr[0][i]]);

printf("arr2[arr[0][%d]] -> %d\n", i, arr2[arr[1][i]]);

printf("arr[0][%d]+1 -> %d\n", i, arr[0][i] + 1);

printf("arr[1][%d]+1 %d\n", i, arr[1][i] + 1);

for (int j = arr[0][i] + 2; j < 11; j++)

{

arr1[j] = arr1[j - 1] + size1;

}

}

}

flag = flag + 1;

if (flag == 5)

{

flag = 1;

}

printf("\n");

}

printf("\nP1\tP2\tP3\n");

for (int i = 0; i < 11; i++)

{

printf("%d\t%d\t%d\n", arr1[i], arr2[i], arr3[i]);

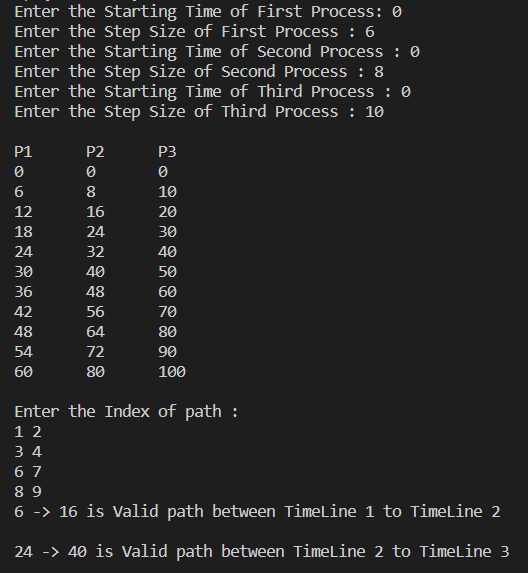
}

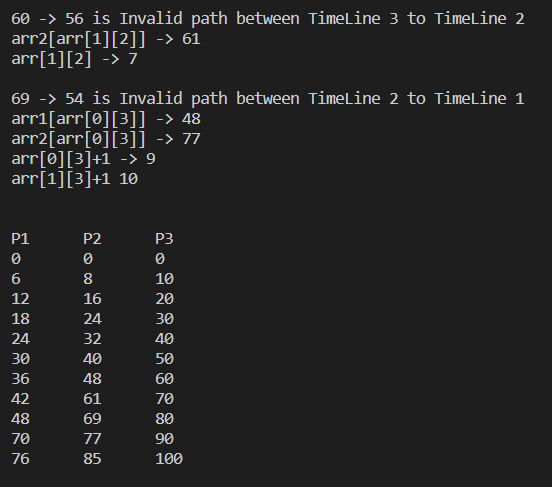
printf("\n");

return 0;

}

# Output





# **IMPLEMENTING LAMPORT’S LOGICAL CLOCK IN ELECTRICITY BILL GENERATOR**

# Problem statement:

Suppose a client wants to pay the bill. So when the client pays the bill, the server should first check whether the client has exceeded the due date or not. If the client has exceeded the due date, he would be penalized 20% of the bill. Then the client is provided with an option to pay the bill. After the successful payment of the bill the client gets the final bill with timestand printed on it.

# Lamport Implementation:

A logical clock is just a counter that holds a number, called timestamp. This number has no relationship to physical time. Its only purpose is to capture the ordering of events. And in order to have this property, the clock needs to satisfy

a→b⟹T(a)<T(b) for any events a and b

Lamport calls it Clock Condition. In other words, each node of a distributed system will have a clock (logical), that is just a counter; and all of the counters store numbers that are in agreement that if something happened the timestamp of this happening will be smaller than the next event.

Each node in a distributed system has a clock, which is a counter that stores the time t that the last event occurred. The counter starts at t = 0.

When an event occurs at the node, t is incremented, that is, t = t + 1.

When a node wants to send a message m to another node, t is incremented and it is sent together with the message, t = t + 1, then, send(m, t).

When a node receives a message, it updates its own clock with the maximum of its current time t and the time t\_received that came with the message and adds one.

That is, t = max(t, t\_received) + 1.

The server first checks the due date of the client from the database.

If payment\_date >= due\_date:

fine = 0.2\*totalBill

totalBill += fine

else:

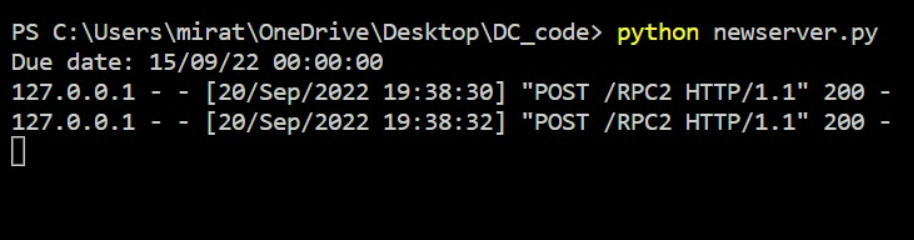
fine = 0

Thus the timestamp is noted at the time of payment and the client is thus provided with the final bill.

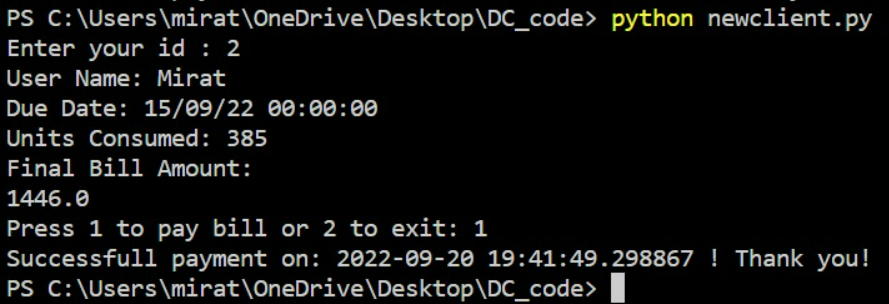
The implementation of the above code can be found here (<https://github.com/Karan-Shah-2513/DC-expt>)

# Output:

**SERVER:**



**CLIENT:**

****

# Conclusion

By performing the above experiment I have understood how to implement a lamport logical clock in a program and understood the following points.

* The usefulness of Lamport Clocks comes from the fact that it can be used to define a total order relation among all events in the system. In Lamport words,
* We can use a system of clocks satisfying the Clock Condition to place a total ordering on the set of all system events. We simply order the events by the times at which they occur. To break ties, we use any arbitrary total ordering of the processes.
* Being able to totally order the events can be very useful in implementing a distributed system. In fact, the reason for implementing a correct system of logical clocks is to obtain such a total ordering.
* The use of Lamport Clocks and total ordering in a distributed system is what let us coordinate or synchronize events of multiple nodes in a way that every node agrees. It is a simple consensus algorithm.

**Thank You**