Vidyavardhini's College of Engineering & Technology

Department of Computer Engineering

Name: Shivam Pandey
Roll No.: 28
Experiment No. 4
Implement Clock Synchronization algorithms
Date of Performance: 15/02/2024
Date of Submission: 22/02/2024

Aim: To implement Clock Synchronization algorithms.

Objective: To implement Clock Synchronization algorithms.

Theory:

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

It follows some simple rules:

1. A process increments its counter before each event in that process;
2. When a process sends a message, it includes its counter value with the message;
3. On receiving a message, the receiver process sets its counter to be the maximum of the message counter and its own counter incremented, before it considers the message received.

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 resynchronizes its logical clock with that sender.

Each process maintains a single Lamport timestamp counter. Each event on the process is tagged with a timestamp from this counter. The counter is incremented before the event timestamp is assigned. If a process has four events, a, b, c, d, they would get Lamport timestamps of 1, 2, 3, 4.

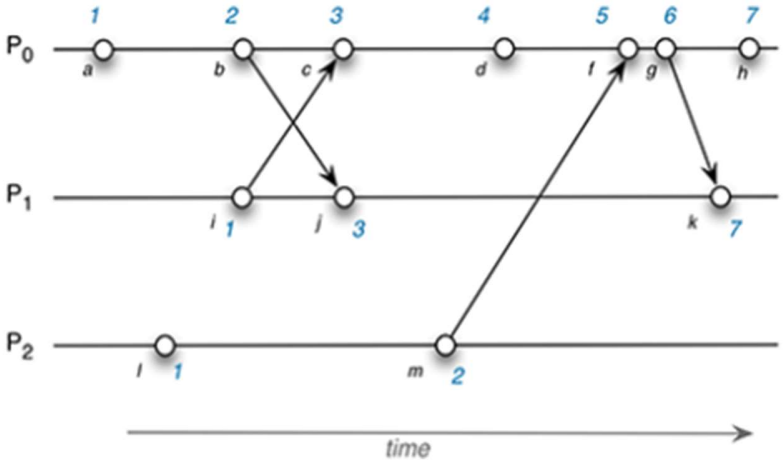
If an event is the sending of a message then the timestamp is sent along with the message.

If an event is the receipt of a message then the algorithm instructs you to compare the current value of the process' timestamp counter (which was just incremented before this event) with the timestamp in the received message. If the timestamp of the received message is greater than that of the current system, the system timestamp is updated with that of the timestamp in the received message plus one. This ensures that the timestamp of the received event and all further timestamps will be greater than that of the timestamp of sending the message as well as all previous messages.

In the figure below, event k in process P_1 is the receipt of the message sent by event g in P_0. If event k was just a normal local event, the P_1 would assign it a timestamp of 4. However, since the received timestamp is 6, which is greater than 4, the timestamp counter is set to 6+1, or 7. Event k gets the timestamp of 7. A local event after k would get a timestamp of 8.

With Lamport timestamps, we're assured that two causally-related events will have timestamps that reflect the order of events. For example, event c happened before event k in the Lamport causal sense: the chain of causal events is c→d, d→f, f→g, and

g→k. Since the happened-before relationship is transitive, we know that c→k (c happened before k). The Lamport timestamps reflect this. The timestamp for c (3) is less than the timestamp for k(7). However, just by looking at timestamps we cannot conclude that there is a causal happened-before relation. For instance, because the timestamp for l (1) is less than the timestamp for j (3) does not mean that l happened before j. Those events happen to be concurrent but we cannot discern that by looking at Lamport timestamps. We need to employ a different technique to be able to make that determination. That technique is the use ofvector timestamps.



Lamport Clock Assignments

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include<bits/stdc++.h>
using namespace std;

int main() {
   int i,j,k; int x=0;
   char a[10][10];
   int n,num[10],b[10][10];
   printf("Enter the no. of physical clocks: ");
   scanf("%d",&n);
   for(i=0;i<n;i++)    {
      printf("\nNo. of nodes for physical clock %d: ",i+1);
```

```
    scanf("%d",&num[i]);
    x=0;
    for(j=0;j<num[i];j++)       {
       printf("\nEnter the name of process: ");
       scanf("%s",&a[i][j]);
       b[i][j]=x + rand() % 10;
       x=b[i][j]+1;

    }

  }
  printf("\n\n");
  for(i=0;i<n;i++)    {
    printf("Physical Clock %d",i+1);
    for(j=0;j<num[i];j++){
       printf("\nProcess %c",a[i][j]);
       printf(" has P.T. : %d ",b[i][j]);

    }
    printf("\n\n");

  }
  x=0;
  for(i=0;i<10;i++)
    for(j=0;j<n;j++)
       for(k=0;k<num[j];k++)
          if(b[j][k]==i)  {
             x = rand() % 10 + x;
             printf("\nLogical Clock Timestamp for process %c",a[j][k]);
             printf(" : %d ",x);
             printf("\n");

          }
  return 0;

}
```

Output:

Enter the no. of physical clocks: 3

No. of nodes for physical clock 1: 2

Enter the name of process: a

Enter the name of process: b

No. of nodes for physical clock 2: 2

Enter the name of process: c

Enter the name of process: d

No. of nodes for physical clock 3: 2

Enter the name of process: e

Enter the name of process: f

Physical Clock 1
Process a has P.T. : 3
Process b has P.T. : 10

Physical Clock 2
Process c has P.T. : 7
Process d has P.T. : 13

Physical Clock 3
Process e has P.T. : 3
Process f has P.T. : 9

Logical Clock Timestamp for process a : 6

Logical Clock Timestamp for process e : 8

Logical Clock Timestamp for process c : 17

Logical Clock Timestamp for process f : 18

Conclusion:

Lamport's algorithm for timestamp assignment is a distributed clock synchronization method in computer science. It ensures ordering and causality in distributed systems by assigning logical timestamps to events. Each process increments its own timestamp when initiating an event and updates it based on messages received from other processes. Lamport timestamps allow for establishing a partial ordering of events in a distributed system, aiding in the analysis of causality and coordination among processes.