BITS Pilani - Hyderabad Campus

Advanced Operating Systems (CG 623)

Lamport's Logical Clocks

Lab Sheet 5

1st Sem 2025-26

Lamport proposed a scheme to order the events in a distributed system by using logical clocks. Due to the absence of synchronized clock and global time in a distributed system, the order in which events occur at two different machines is impossible to be determined based on the local time at which they occurred.

We will work on the following example by using the following concepts.
1) Happened Before Relationship
2) Logical clocks

For any two events a & b, a is said to happened before b is denoted as a -> b, if they are in same process.
If events occur at different processes then for any message (m)
Send (m) -> Receive (m)
If a -> b and b->c then a ->c i.e. -> is transitive.
If events casually affect each other then they are said to be casually related events

$$a \rightarrow b$$

Two events are concurrent if a !-> b and !b -> a i.e. a || b
Conditions satisfied by Logical Clocks.
a) For any two events occurring on same process,

$$a \rightarrow b \text{ if } Ci(a) < Ci(b).$$

b) Clock Ci is implemented between any two events of the same process as

$$Ci = Ci + d \ (d>0)$$

c) If event a is sending message by process Pi and is received by process Pj, then
tm = Ci(a), and Cj = max(Cj+d, tm), d>0

In the following code, there are two processes where process1 sends a message to process 2, and process 2 updates its timestamp according to that.

(**Program 1**)

```
#include<stdio.h>
void main()
{
        int p1[10],p2[10];
        int e,i,m1,m2;
        printf("\nenter the no of events in p1 & p2\n");
        scanf("%d",&e);
        printf("\n enter the event of p1 which will send
        msg\n"); scanf("%d",&m1);
        printf("\nenter the event of p2 which will receive the msg\n");
        scanf("%d",&m2);
        p1[0] = 0;
        p2[0] = 0;
```

```
        for(i=1;i<e;i++)
        {
                if(i==m2)
                {
                        if(m1 > p2[i-1])
                                p2[i]=m1+1;
                        else
                                p2[i]=p2[i-1]+1;
                        p1[i]=p1[i-1]+1;
                }
                else
                {
                        p1[i]=p1[i-1]+1;
                        p2[i]=p2[i-1]+1;

                }
        }
        printf("\n Time stamp for P1\n");
        for(i=0;i<e;i++)
        {
                printf("%d ", p1[i]);
        }
        printf("\n Time stamp for P2\n");
        for(i=0;i<e;i++)
        {
        printf("%d ",p2[i]);
        }
        printf("\n");
}
```

```
iot@iot-Satellite-U840:~$ ./t1

enter the no of events in p1 & p2
8

enter the event of p1 which will send msg
5

enter the event of p2 which will recieve the msg
3

Time stamp for P1
0 1 2 3 4 5 6 7
Time stamp for P2
0 1 2 6 7 8 9 10
iot@iot-Satellite-U840:~$
```

Task: You can use the above program as a reference and write a program for atleast 3 processes, where any process can send messages to any other process. Show the time stamp for each process.

Another way of Implementation using Pipes as Communication medium between threads:

Program2:

```
/*LAMPORT'S LOGICAL CLOCK USING PIPES*/

#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

#define THREAD_COUNT 3

int fd1[2], fd2[2], fd3[2];   //File descriptors

int max(int a, int b)    //Returns maximum value among two integers
{
   if(a>b)
     return a;
   else
     return b;
}

/*****PROCESS 1*****/
void *process_1_func(void *arg)
{
     int p1[4];
     int i,timestamp;
     char msg[4];

     p1[0]=0;  //initial clock value

     ++p1[0]; //Incrementing clock value of local event 1 before
writing

     snprintf(msg,sizeof(msg),"%d",p1[0]);

    //Process 1 is writing to Process 2(P1->P2)
    if(write(fd1[1],msg,1)==0)
    {
       printf("\nError in write");
       exit(1);
    }

    //Assigning clock values to local events 2 and 3
    p1[1]=p1[0]+1;
    p1[2]=p1[1]+1;


    //Process 1 event 4 is reading
     if(read(fd2[0],msg, 1)==0)
```

```
     {
       printf("\nError in read");
       exit(1);
     }
     else
     {
       timestamp=atoi(msg);
       p1[3]= max(p1[2]+1, timestamp+1);
       printf("\nMessage received: Time stamp at event 4(P1)
       =%d",p1[3]);
     }


    //Printing timestamps of events of process 1
      for(i=0;i<4;i++)
         printf("\nPROCESS 1: Timestamp at event %d = %d ",i+1,p1[i]);

     pthread_exit(NULL);
}

/*****PROCESS 2*****/
void *process_2_func(void *arg)
{

    int p2[2];
    int i,timestamp;
    char msg[4];

    p2[0]=0;  //initial clock value

    //Process 2 event 1 is reading
     if(read(fd1[0],&msg, 1)==0)
     {
           printf("\nError in read");
           exit(1);
     }
     else
     {
        timestamp=atoi(msg);
        p2[0] = max(p2[0]+1, timestamp+1);
        printf("\nMessage Received: Time stamp at event 1(P2) =
        %d",p2[0]);

     }

     p2[1]=p2[0]+1;   //Incrementing clock value of local event 2
     before writing

      snprintf(msg,sizeof(msg),"%d",p2[1]);

    //Process 2 event 2 is writing to Process 3 (P2->P3)
    if(write(fd3[1],msg,1)==0)
    {
       printf("\nError in write");
       exit(1);
    }
```

```
   //Printing timestamps of events of process 2
   for(i=0;i<2;i++)
     printf("\nPROCESS 2: Timestamp at event %d = %d",i+1,p2[i]);

   pthread_exit(NULL);
}

/*****PROCESS 3*****/
void *process_3_func(void *arg)
{
    int p3[4];
    int i,timestamp;
    char msg[4];

    p3[0]=0;    //initial timestamp
    ++(p3[0]);  //Assigning timestamp to local event 1 to P3

    p3[1]=p3[0]+1;   //Incrementing clock value of event 2 before
writing

    snprintf(msg,sizeof(msg),"%d",p3[1]);

    //Process 3 event 2 is writing to Process 1(P3->P1)
    if(write(fd2[1],msg,1)==0)
    {
       printf("\nError in write");
       exit(1);
    }

    //Assigning timestamp to local event 3 of process 3
    p3[2]=p3[1]+1;


   //Process 3 event 4 is reading
     if(read(fd3[0],msg, 1)==0)
     {
      printf("\nError in read");
      exit(1);
     }
     else
     {
       timestamp=atoi(msg);
       p3[3] = max(p3[2]+1, timestamp+1);
       printf("\nMessage Received: Time stamp at event 4(P3) =
%d",p3[3]);
     }

  //Printing timestamps of events of process 3
  for(i=0;i<4;i++)
     printf("\nPROCESS 3: Timestamp at event %d = %d",i+1,p3[i]);
```

```
  pthread_exit(NULL);
}

int main(int argc, char **argv)
{
   pthread_t thr[THREAD_COUNT];
   int i, rc;

  /*Pipes Creation*/

      if(pipe(fd1)<0)                  //Pipe 1 (P1-P2)
        printf("\nError opening pipe 1");
      if(pipe(fd2)<0)                  //Pipe 2 (P3-P1)
          printf("\nError opening pipe 2");
      if(pipe(fd3)<0)                  //Pipe 3 (P2-P3)
        printf("\nError opening pipe 3");

   /*Creating 3 threads simulating 3 processes*/

       if((rc = pthread_create(&thr[0], NULL, process_1_func, NULL)))
       {
          fprintf(stderr, "ERROR: pthread_create(), rc: %d\n", rc);
          return EXIT_FAILURE;
       }

       if((rc = pthread_create(&thr[1], NULL, process_2_func, NULL)))
       {
          fprintf(stderr, "ERROR: pthread_create(), rc: %d\n", rc);
          return EXIT_FAILURE;
       }

       if((rc = pthread_create(&thr[2], NULL, process_3_func, NULL)))
       {
          fprintf(stderr, "ERROR: pthread_create(), rc: %d\n", rc);
          return EXIT_FAILURE;
       }



      pthread_join(thr[0],NULL);
      pthread_join(thr[1],NULL);
      pthread_join(thr[2],NULL);

   return EXIT_SUCCESS;
}
```

/*OUTPUT

```
$ gcc -pthread -o lamp lamp.c
$ ./lamp
```

```
Message received: Time stamp at event 4(P1) = 4
PROCESS 1: Timestamp at event 1 = 1
PROCESS 1: Timestamp at event 2 = 2
PROCESS 1: Timestamp at event 3 = 3
PROCESS 1: Timestamp at event 4 = 4
Message Received: Time stamp at event 1(P2) = 2
PROCESS 2: Timestamp at event 1 = 2
PROCESS 2: Timestamp at event 2 = 3
Message Received: Time stamp at event 4(P3) = 4
PROCESS 3: Timestamp at event 1 = 1
PROCESS 3: Timestamp at event 2 = 2
PROCESS 3: Timestamp at event 3 = 3
PROCESS 3: Timestamp at event 4 = 4

*/
```

Task: Draw a communication diagram showing the messages and their clock values.
