**Department of Computer Engineering**

**Academic Term: January - May 2021**

**Class : *B.E (Computer) Semester VIII***

**Subject Name: *Distributed Computing***

**Subject Code : *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **4** |
| **Title:** | **Lamport Timestamp** |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of the Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher :**

**Code:**

from multiprocessing import Process, Pipe

from os import getpid

from datetime import datetime

def local\_time(counter):

    return ' (LAMPORT\_TIME={}, LOCAL\_TIME={})'.format(counter, datetime.now())

def calc\_recv\_timestamp(recv\_time\_stamp, counter):

    return max(recv\_time\_stamp, counter) + 1

def event(pid, counter):

    counter += 1

    print('Something happened in {} !'.format(pid) + local\_time(counter))

    return counter

def send\_message(pipe, pid, counter):

    counter += 1

    pipe.send(('Empty shell', counter))

    print('Message sent from ' + str(pid) + local\_time(counter))

    return counter

def recv\_message(pipe, pid, counter):

    message, timestamp = pipe.recv()

    counter = calc\_recv\_timestamp(timestamp, counter)

    print('Message received at ' + str(pid)  + local\_time(counter))

    return counter

def process\_one(pipe12):

    pid = getpid()

    counter = 0

    print("Process 1 Init Counter: "+str(counter))

    counter = event(pid, counter)

    print("Process 1 Counter: "+str(counter))

    counter = send\_message(pipe12, pid, counter)

    print("Process 1 Counter: "+str(counter))

    counter  = event(pid, counter)

    print("Process 1 Counter: "+str(counter))

    counter = recv\_message(pipe12, pid, counter)

    print("Process 1 Counter: "+str(counter))

    counter  = event(pid, counter)

    print("Process 1 Counter: "+str(counter))

def process\_two(pipe21, pipe23):

    pid = getpid()

    counter = 0

    print("Process 2 Init Counter: "+str(counter))

    counter = recv\_message(pipe21, pid, counter)

    print("Process 2 Counter: "+str(counter))

    counter = send\_message(pipe21, pid, counter)

    print("Process 2 Counter: "+str(counter))

    counter = send\_message(pipe23, pid, counter)

    print("Process 2 Counter: "+str(counter))

    counter = recv\_message(pipe23, pid, counter)

    print("Process 2 Counter: "+str(counter))

def process\_three(pipe32):

    pid = getpid()

    counter = 0

    print("Process 3 Init Counter: "+str(counter))

    counter = recv\_message(pipe32, pid, counter)

    print("Process 3 Counter: "+str(counter))

    counter = send\_message(pipe32, pid, counter)

    print("Process 3 Counter: "+str(counter))

if \_\_name\_\_ == '\_\_main\_\_':

    oneandtwo, twoandone = Pipe()

    twoandthree, threeandtwo = Pipe()

    process1 = Process(target=process\_one, args=(oneandtwo,))

    process2 = Process(target=process\_two, args=(twoandone, twoandthree))

    process3 = Process(target=process\_three, args=(threeandtwo,))

    process1.start()

    process2.start()

    process3.start()

    process1.join()

    process2.join()

    process3.join()

**Output:**

Process 2 Init Counter: 0

Process 1 Init Counter: 0

Something happened in 15248 ! (LAMPORT\_TIME=1, LOCAL\_TIME=2021-04-19 11:27:55.446930)

Process 1 Counter: 1

Message sent from 15248 (LAMPORT\_TIME=2, LOCAL\_TIME=2021-04-19 11:27:55.446930)

Message received at 3288 (LAMPORT\_TIME=3, LOCAL\_TIME=2021-04-19 11:27:55.446930)

Process 1 Counter: 2

Process 2 Counter: 3

Something happened in 15248 ! (LAMPORT\_TIME=3, LOCAL\_TIME=2021-04-19 11:27:55.446930)

Message sent from 3288 (LAMPORT\_TIME=4, LOCAL\_TIME=2021-04-19 11:27:55.456176)

Process 1 Counter: 3

Process 2 Counter: 4

Message received at 15248 (LAMPORT\_TIME=5, LOCAL\_TIME=2021-04-19 11:27:55.456176)

Process 1 Counter: 5

Process 3 Init Counter: 0

Message sent from 3288 (LAMPORT\_TIME=5, LOCAL\_TIME=2021-04-19 11:27:55.456176)

Something happened in 15248 ! (LAMPORT\_TIME=6, LOCAL\_TIME=2021-04-19 11:27:55.456176)

Message received at 5060 (LAMPORT\_TIME=6, LOCAL\_TIME=2021-04-19 11:27:55.457175)

Process 2 Counter: 5

Process 1 Counter: 6

Process 3 Counter: 6

Message sent from 5060 (LAMPORT\_TIME=7, LOCAL\_TIME=2021-04-19 11:27:55.458175)

Message received at 3288 (LAMPORT\_TIME=8, LOCAL\_TIME=2021-04-19 11:27:55.458175)

Process 3 Counter: 7

Process 2 Counter: 8

**Conclusion:**

A Lamport logical clock is an incrementing counter maintained in each process. Conceptually, this logical clock can be thought of as a clock that only has meaning concerning messages moving between processes. When a process receives a message, it resynchronizes its logical clock with that sender (causality).

**Post Lab Assignments**

1. What’s the relationship between the events described in Lamport’s algorithm?

Ans: -

Lamport Algorithm: Lamport developed a “happens-before” notation to express this: If a and b are events in the same process, and a occurs before b, then a → b is true. If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then a → b. This relationship is transitive i.e. a→b and b→c then a→c. Satisfying conditions for implementing clock: If a→b then c(a)<c(b). Implementation of logical clock: Condition 1: If a and b are two events within the same process Pi and a occur before b then Ci(a) < Ci (b).

Condition 2: if a is the sending of a message by process Pi and b is the receipt of that message by process Pj then Ci(a) < Cj(b).

Condition 3: A clock Ci associated with a process Pi must always go forward never backward that is correction to the time of clock is done by +ive adding.

1. What are Partial Ordering and Total Ordering (Give examples) of events

Ans: -

Total ordering is an ordering that defines the exact order of every element in the series.

Partial ordering of elements in a series is an ordering that doesn't specify the exact order of every item, but only defines the order between certain key items that depend on each other.

The meaning of these words is the same in the context of distributed computing. The only significance of distributed computing to these terms is the fact that partial ordering of events is much commoner than total ordering. In a local, single-threaded application, the order in which events happen is ordered, implicitly, since the CPU can only do one thing at a time. In a distributed system, you generally only coordinate a partial ordering of those events that have a dependency on one another and let other events happen in whatever order they happen.

Example, taken from the comments: If you have three events {A, B, C}, then they are ordered if they always have to happen in the order A > B > C. However, if A must happen before C, but you don't care when B happens, then they are partially ordered. In this case we would say that the sequences A > B > C, A > C > B, and B > A > C all satisfy the partial ordering

**Department of Computer Engineering**

**Academic Term: January - May 2021**

**Class : *B.E (Computer) Semester VIII***

**Subject Name: *Distributed Computing***

**Subject Code : *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **5** |
| **Title:** | **Vector Timestamp** |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of the Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher :**

**Code:**

from multiprocessing import Process, Pipe

from os import getpid

from datetime import datetime

from time import sleep

def local\_time(counter):

    return " (VECTOR\_TIME={}, LOCAL\_TIME={})".format(counter, datetime.now())

def event(pid, counter):

    counter[pid] += 1

    print("Something happened in {} !".format(pid) + local\_time(counter))

    return counter

def calc\_recv\_timestamp(recv\_time\_stamp, counter):

    for id in range(len(counter)):

        counter[id] = max(recv\_time\_stamp[id], counter[id])

    return counter

def send\_message(pipe, pid, counter):

    counter[pid] += 1

    pipe.send(("Empty shell", counter))

    print("Message sent from " + str(pid) + local\_time(counter))

    return counter

def recv\_message(pipe, pid, counter):

    message, timestamp = pipe.recv()

    counter = calc\_recv\_timestamp(timestamp, counter)

    counter[pid] += 1

    print("Message received at " + str(pid) + local\_time(counter))

    return counter

def process\_one(pipe12, pipe13):

    pid = 0

    counter = [0, 0, 0]

    counter = event(pid, counter)

    counter = send\_message(pipe12, pid, counter)

    counter = event(pid, counter)

    counter = recv\_message(pipe12, pid, counter)

    counter = event(pid, counter)

def process\_two(pipe21, pipe23):

    pid = 1

    counter = [0, 0, 0]

    counter = event(pid, counter)

    counter = recv\_message(pipe21, pid, counter)

    counter = recv\_message(pipe23, pid, counter)

    counter = send\_message(pipe21, pid, counter)

    counter = event(pid, counter)

def process\_three(pipe32, pipe31):

    pid = 2

    counter = [0, 0, 0]

    counter = send\_message(pipe32, pid, counter)

    counter = event(pid, counter)

if \_\_name\_\_ == "\_\_main\_\_":

    oneandtwo, twoandone = Pipe()

    twoandthree, threeandtwo = Pipe()

    oneandthree, threeandone = Pipe()

    process1 = Process(target=process\_one, args=(oneandtwo, oneandthree))

    process2 = Process(target=process\_two, args=(twoandone, twoandthree))

    process3 = Process(target=process\_three, args=(threeandtwo, threeandone))

    process1.start()

    process2.start()

    process3.start()

    process1.join()

    process2.join()

    process3.join()

**Output:**

# Something happened in 0 ! (VECTOR\_TIME=[1, 0, 0], LOCAL\_TIME=2021-03-25 20:43:32.215746)

# Something happened in 1 ! (VECTOR\_TIME=[0, 1, 0], LOCAL\_TIME=2021-03-25 20:43:32.215778)

# Message sent from 0 (VECTOR\_TIME=[2, 0, 0], LOCAL\_TIME=2021-03-25 20:43:32.215812)

# Something happened in 0 ! (VECTOR\_TIME=[3, 0, 0], LOCAL\_TIME=2021-03-25 20:43:32.215819)

# Message received at 1 (VECTOR\_TIME=[2, 2, 0], LOCAL\_TIME=2021-03-25 20:43:32.215836)

# Message sent from 2 (VECTOR\_TIME=[0, 0, 1], LOCAL\_TIME=2021-03-25 20:43:32.216401)

# Message received at 1 (VECTOR\_TIME=[2, 3, 1], LOCAL\_TIME=2021-03-25 20:43:32.216416)

# Something happened in 2 ! (VECTOR\_TIME=[0, 0, 2], LOCAL\_TIME=2021-03-25 20:43:32.216421)

# Message sent from 1 (VECTOR\_TIME=[2, 4, 1], LOCAL\_TIME=2021-03-25 20:43:32.216440)

# Something happened in 1 ! (VECTOR\_TIME=[2, 5, 1], LOCAL\_TIME=2021-03-25 20:43:32.216445)

# Message received at 0 (VECTOR\_TIME=[4, 4, 1], LOCAL\_TIME=2021-03-25 20:43:32.216479)

# Something happened in 0 ! (VECTOR\_TIME=[5, 4, 1], LOCAL\_TIME=2021-03-25 20:43:32.216492)

**Conclusion:**

Through this experiment we can observe that through Vector timestamp concurrency is achieved and the process of synchronizations for different processes becomes much more efficient, making it local time independent of the servers.

**Post Lab Assignments**

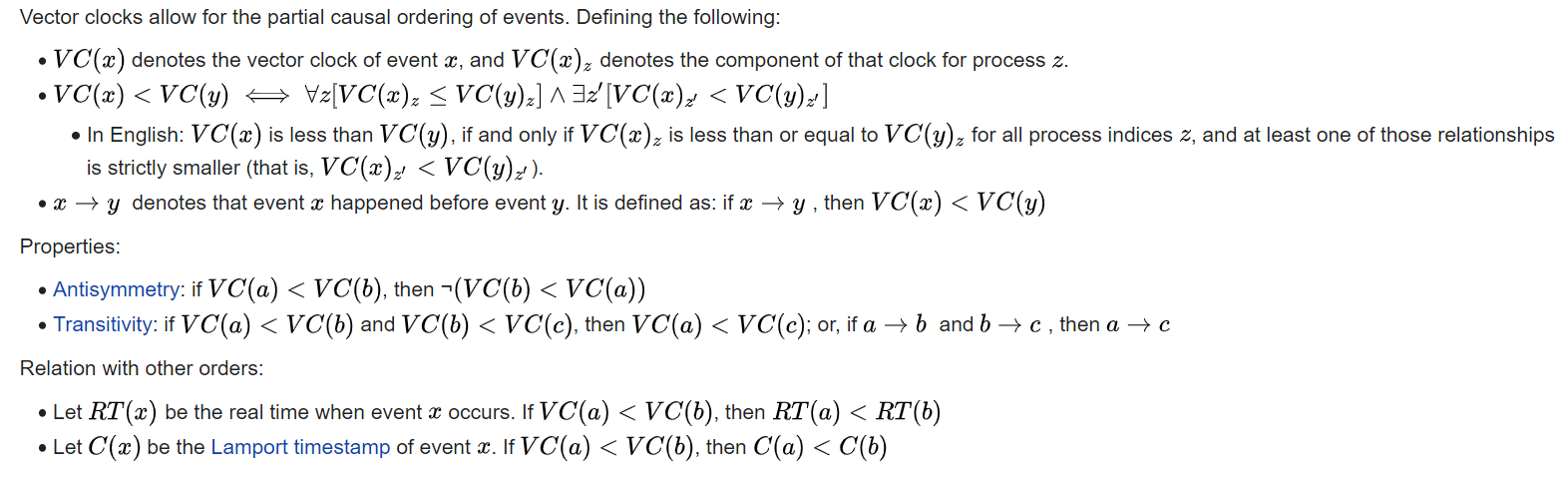
1. Differences between Lamport and Vector timestamps

Ans:-

Although similar they have different purposes: Vector Timestamp can distinguish whether two operations are concurrent or one is causally dependent on the other; Lamport timestamps enforces total ordering. Total ordering although more compact cannot tell whether two operations are concurrent or causally dependent.

1. What is the relation of events maintained in Vector timestamps?

Ans:-



**Department of Computer Engineering**

**Academic Term: January - May 2021**

**Class : *B.E (Computer) Semester VIII***

**Subject Name: *Distributed Computing***

**Subject Code : *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **6** |
| **Title:** | Election Algorithm (Bully Algorithm) |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of the Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher :**

**Code:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct process {

   int no;

   int priority;

   int active;

   struct process \*next;

};

typedef struct process proc;

struct priority {

   int pri;

   struct priority \*next;

   proc \*pp;

};

typedef struct priority pri;

pri\* find\_priority(proc \*head, pri \*head1) {

   proc \*p1;

   pri \*p2, \*p3;

   p1 = head;

   while (p1->next != head) {

      if (p1->active == 1) {

         if (head1 == NULL) {

            head1 = (pri\*) malloc(sizeof(pri));

            head1->pri = p1->priority;

            head1->next = NULL;

            head1->pp = p1;

            p2 = head1;

         } else {

            p3 = (pri\*) malloc(sizeof(pri));

            p3->pri = p1->priority;

            p3->pp = p1;

            p3->next = NULL;

            p2->next = p3;

            p2 = p2->next;

         }

         p1 = p1->next;

      } else

         p1 = p1->next;

   } //end while

   p3 = (pri\*) malloc(sizeof(pri));

   p3->pri = p1->priority;

   p3->pp = p1;

   p3->next = NULL;

   p2->next = p3;

   p2 = p2->next;

   p3 = head1;

   return head1;

} //end find\_priority()

int find\_max\_priority(pri \*head) {

   pri \*p1;

   int max = -1;

   int i = 0;

   p1 = head;

   while (p1 != NULL) {

      if (max < p1->pri && p1->pp->active == 1) {

         max = p1->pri;

         i = p1->pp->no;

      }

      p1 = p1->next;

   }

   return i;

}

void bully() {

   proc \*head;

   proc \*p1;

   proc \*p2;

   int n, i, pr, maxpri, a, pid, max, o;

   char ch;

   head = p1 = p2 = NULL;

   printf("\nnEnter how many process: ");

   scanf("%d", &n);

   for (i = 0; i < n; i++) {

      printf("\nEnter priority of process %d: ", i + 1);

      scanf("%d", &pr);

      printf("\nIs process with id %d is active ?(0/1) :", i + 1);

      scanf("%d", &a);

   if (head == NULL) {

         head = (proc\*) malloc(sizeof(proc));

         if (head == NULL) {

            printf("\nMemory cannot be allocated");

            getch();

            exit(0);

         }

         head->no = i + 1;

         head->priority = pr;

         head->active = a;

         head->next = head;

         p1 = head;

      } else {

         p2 = (proc\*) malloc(sizeof(proc));

         if (p2 == NULL) {

            printf("\nMemory cannot be allocated");

            getch();

            exit(0);

         }

         p2->no = i + 1;

         p2->priority = pr;

         p2->active = a;

         p1->next = p2;

         p2->next = head;

         p1 = p2;

      }

   } //end for

   printf("\nEnter the process id that invokes election algorithm: ");

   scanf("%d", &pid);

   p2 = head;

   while (p2->next != head) {

      if (p2->no == pid) {

         p2 = p2->next;

         break;

      }

      p2 = p2->next;

   }

   printf("\nProcess with id %d has invoked election algorithm", pid);

   printf("\t\nElection message is sent to processes");

   while (p2->next != head) {

      if (p2->no > pid)

         printf("%d", p2->no);

      p2 = p2->next;

   }

   printf("%d", p2->no);

   p2 = head;

   max = 0;

   while (1) {

      if (p2->priority > max && p2->active == 1)

         max = p2->no;

      p2 = p2->next;

      if (p2 == head)

         break;

   }

   printf("\n\tProcess with the id %d is the co-ordinator", max);

   while (1) {

      printf("\nDo you want to continue?(y/n): ");

      scanf("%c", &ch);

      if (ch == 'n' || ch == 'N')

         break;

      p2 = head;

      while (1) {

         printf("\nEnter the process with id %d is active or not (0/1): ",

               p2->no);

         scanf("%d", &p2->active);

         p2 = p2->next;

         if (p2 == head)

            break;

      }

      printf("\nEnter the process id that invokes election algorithm: ");

      scanf("%d", &pid);

      printf("\n\tElection message is sent to processes ");

      while (p2->next != head) {

         if (p2->no > pid)

            printf("%d", p2->no);

         p2 = p2->next;

      }

      printf("%d", p2->no);

      p2 = head;

      max = 0;

      while (1) {

         if (p2->no > max && p2->active == 1)

            max = p2->no;

         p2 = p2->next;

         if (p2 == head)

            break;

      }

      printf("\n\tProcess with id %d is the co-ordinator", max);

   }

}

void main() {

   bully();

   getch();

}

**Output:**

|  |  |
| --- | --- |
|  | nEnter how many process: 6 |
|  |  |
|  | Enter priority of process 1: 1 |
|  |  |
|  | Is process with id 1 is active ?(0/1) :1 |
|  |  |
|  | Enter priority of process 2: 2 |
|  |  |
|  | Is process with id 2 is active ?(0/1) :1 |
|  |  |
|  | Enter priority of process 3: 3 |
|  |  |
|  | Is process with id 3 is active ?(0/1) :1 |
|  |  |
|  | Enter priority of process 4: 4 |
|  |  |
|  | Is process with id 4 is active ?(0/1) :1 |
|  |  |
|  | Enter priority of process 5: 5 |
|  |  |
|  | Is process with id 5 is active ?(0/1) :1 |
|  |  |
|  | Enter priority of process 6: 6 |
|  |  |
|  | Is process with id 6 is active ?(0/1) :0 |
|  |  |
|  | Enter the process id that invokes election algorithm: 4 |
|  |  |
|  | Process with id 4 has invoked election algorithm |
|  | Election message is sent to processes56 |
|  | Process with the id 5 is the co-ordinator |
|  | Do you want to continue?(y/n): n |
|  | \*/ |

**Conclusion:**

The bully Algorithm proposed by Garcia-Molina follows the following algorithm

• When a process notices that the coordinator is no longer responding to requests, it initiates an election.

• A process, P, holds an election as follows:

1. P sends an ELECTION message to all processes with higher numbers

2. If no one responds, P wins the election and becomes the coordinator

3. If one of the higher-up’s answers, it takes over. P's job is done

• If a process can get an ELECTION message from one of its lower-numbered colleagues.

The message arrives => the receiver sends an OK message back to the sender to indicate that he is alive and will take over.

The receiver then holds an election, unless it is already holding one.

• Eventually, all processes give up but one does not give up and that one is the new coordinator.

It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.

• If a process that previously down came back up, it holds an election. ♣ If it happens to be the highest-numbered process currently running, it will win the election and take over the coordinator's job.

• Thus, the biggest guy in town always wins, hence the name "bully algorithm."

**Post Lab Assignments**

1. Who initiates the Election Process?

Ans: -

The network nodes communicate among themselves to decide which of them will get into the "coordinator" state. For that, they need some method to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the coordinator.

1. What is the significance of a coordinator in a system?

Ans: -

To perform coordination, distributed systems employ the concept of coordinators. The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.