**DISTRIBUTED COMPUTING EXPERIMENT 4**

**AIM:**

Clock Synchronization algorithms

**CODE:**

1. **Vector Timestamp**

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:**

# VECTOR 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)

1. **Lamport Algorithm**

#include <stdio.h>

int max1(int a, int b)

{

if (a > b)

return a;

else

return b;

}

void display(int e1, int e2,

int p1[5], int p2[3])

{

int i;

printf("\nThe time stamps of "

"events in P1:\n");

for (i = 0; i < e1; i++) {

printf("%d ", p1[i]);

}

printf("\nThe time stamps of "

"events in P2:\n");

// Print the array p2[]

for (i = 0; i < e2; i++)

printf("%d ", p2[i]);

}

void lamportLogicalClock(int e1, int e2,

int m[5][3])

{

int i, j, k, p1[e1], p2[e2];

for (i = 0; i < e1; i++)

p1[i] = i + 1;

for (i = 0; i < e2; i++)

p2[i] = i + 1;

for (i = 0; i < e2; i++)

printf("\te2%d", i + 1);

for (i = 0; i < e1; i++) {

printf("\n e1%d \t", i + 1);

for (j = 0; j < e2; j++)

printf("%d\t", m[i][j]);

}

for (i = 0; i < e1; i++) {

for (j = 0; j < e2; j++) {

// Change the timestamp if the

// message is sent

if (m[i][j] == 1) {

p2[j] = max1(p2[j], p1[i] + 1);

for (k = j + 1; k < e2; k++)

p2[k] = p2[k - 1] + 1;

}

// Change the timestamp if the

// message is reeived

if (m[i][j] == -1) {

p1[i] = max1(p1[i], p2[j] + 1);

for (k = i + 1; k < e1; k++)

p1[k] = p1[k - 1] + 1;

}

}

}

display(e1, e2, p1, p2);

}

int main()

{

int e1 = 5, e2 = 3, m[5][3];

m[0][0] = 0;

m[0][1] = 0;

m[0][2] = 0;

m[1][0] = 0;

m[1][1] = 0;

m[1][2] = 1;

m[2][0] = 0;

m[2][1] = 0;

m[2][2] = 0;

m[3][0] = 0;

m[3][1] = 0;

m[3][2] = 0;

m[4][0] = 0;

m[4][1] = -1;

m[4][2] = 0;

lamportLogicalClock(e1, e2, m);

return 0;

}

/\*

Output

e21 e22 e23

e11 0 0 0

e12 0 0 1

e13 0 0 0

e14 0 0 0

e15 0 -1 0

The time stamps of events in P1:

1 2 3 4 5

The time stamps of events in P2:

1 2 3

\*/

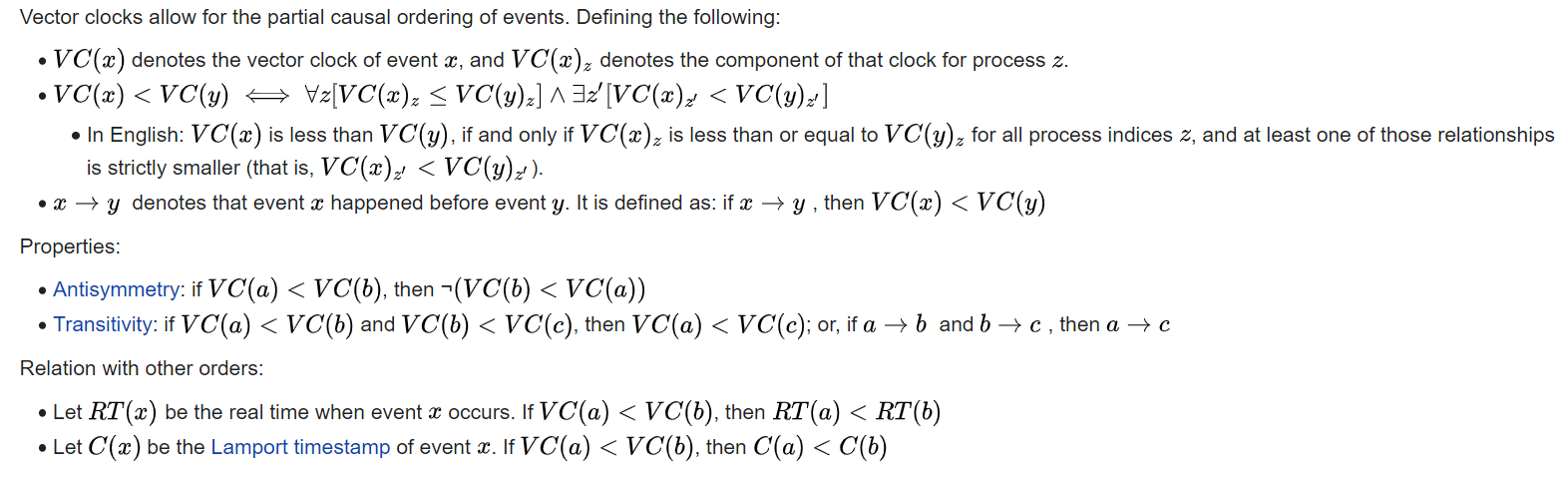
**CONCLUSION:**

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:**

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

Ans: -



2. What is Partial Order and Total Order (Give examples)

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 exactly 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 totally 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 totally 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

3. 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.