

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PURWANCHAL CAMPUS

A

LAB REPORT

ON

“**Simulation for Clock Synchronization in Distributed System**

**using Vector Clock**”

**Submitted By**:

SOHEL AKHTAR (075 BCT 081)

**Submitted To**:

PUKAR KARKI

**DEPARTMENT OF**

**ELECTRONICS AND COMPUTER ENGINEERING**

**DHARAN, NEPAL**

**Lab 2: Simulation for Clock Synchronization in Distributed System using Vector Clock.**

**Vector clocks:** Mattern and Fidge developed vector clocks to overcome the shortcoming of Lamport’s clocks: the fact that from L(e) < L(e′) we cannot conclude that e → e′. A vector clock for a system of N processes is an array of N integers. Each process keeps its own vector clock Vi, which it uses to timestamp local events. Like Lamport timestamps, processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks:

VC1: Initially, **Vi[j]** = 0, for i, j = 1,2… N.

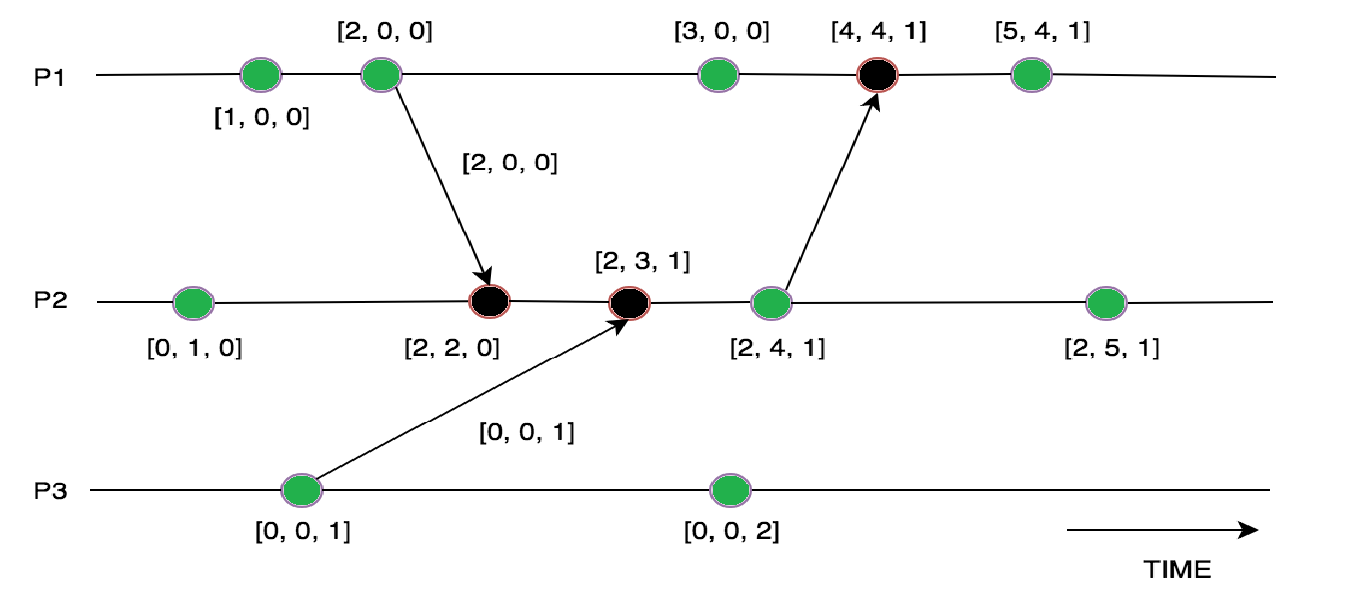
VC2: Just before **pi** timestamps an event, it sets **Vi[i]:= Vi[i] + 1.**

VC3: **pi** includes the value t = Vi in every message it sends.

VC4: When **pi** receives a timestamp t in a message, it sets **Vi[j]:= max(Vi[j], t[j])**, for j = 1,2...N.

Taking the component-wise maximum of two vector timestamps in this way is known as a merge operation.

**Vector Timestamps for the events**



We may compare vector timestamps as follows:

**V = V′ iff V[j] = V′[j] for j = 1, 2… ,N**

**V ≤ V′ iff V[j ] ≤ V′[j] for j = 1, 2… ,N**

**V < V′ iff V ≤ V′ ∧ V ≠ V′**

Vector timestamps have the disadvantage, compared with Lamport timestamps, of taking up an amount of storage and message payload that is proportional to N, the number of processes.

**Source Code**

import pprint

processList = []

logicalClock = {}

timeStamp = {}

processIndexAdd = {}

def addProcess():

    pName = input("Enter 3 Processes Name seperated by space: ")

    processList = pName.split()

    for process in processList:

        logicalClock[process] = (0, 0, 0)

    for i in range(0, len(processList)):

        dummy\_list = [0, 0, 0]

        dummy\_list[i] = 1

        processIndexAdd[processList[i]] = tuple(dummy\_list)

    print(logicalClock, processIndexAdd, sep="\n")

# tuple(map(sum, zip(a, b)))

def addEvent():

    pName = input("Enter the Process for which you want to add an event: ")

    eName = input('Enter Event name: ')

    etype = input("Enter the event type(normal/message): ")

    if etype == "normal":

        logicalClock[pName] = tuple(

            map(sum, zip(logicalClock[pName], processIndexAdd[pName])))

        timeStamp[eName] = logicalClock[pName]

    if etype == "message":

        logicalClock[pName] = tuple(

            map(sum, zip(logicalClock[pName], processIndexAdd[pName])))

        timeStamp[eName] = logicalClock[pName]

        sendmessage(timeStamp[eName])

def sendmessage(t):

    eName = input("Enter the Event which will receive the message: ")

    pName = input("Enter the process on which this event will occur: ")

    dummyList = list(logicalClock[pName])

    for i in range(3):

        if t[i] > logicalClock[pName][i]:

            dummyList[i] = t[i]

    logicalClock[pName] = tuple(

        map(sum, zip(dummyList, processIndexAdd[pName])))

    timeStamp[eName] = logicalClock[pName]

def display():

    pprint.pprint(timeStamp)

if \_\_name\_\_ == "\_\_main\_\_":

    addProcess()

    while(1):

        print("\n1.ADD EVENT\n2.DISPLAY TIMESTAMP\n3.EXIT")

        n = int(input("Enter your choice: "))

        if n == 1:

            addEvent()

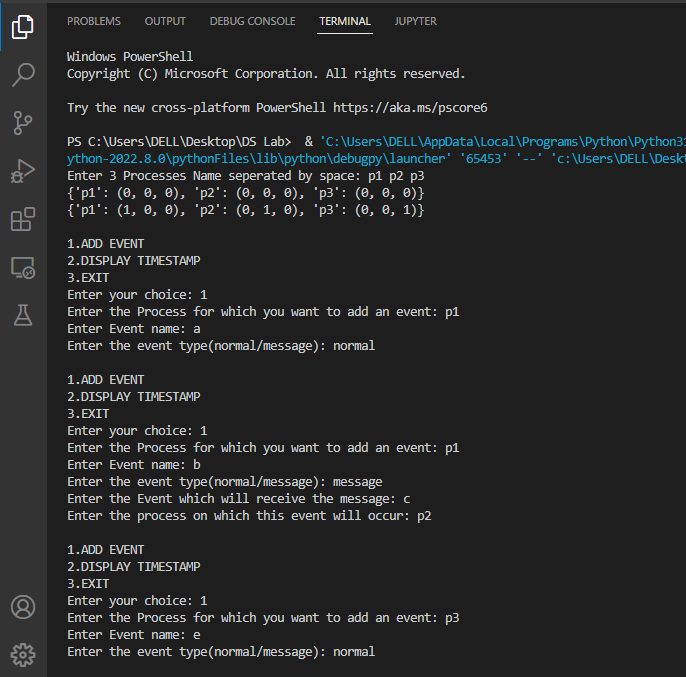
        elif n == 2:

            display()

        else:

            break

**Output**

****

