CSE 5306 DISTRIBUTED SYSTEMS – 004

PROJECT-1

MEMBERS:

- Vignesh Manikandan – 1002012757
- Shane Sam Antony Rebinto Sam – 1002080770

I have neither given nor received unauthorized assistance on this work. I will not post the project description and the solution online.

Sign: Vignesh Manikandan Date: 03/23/2023
Shane Sam Antony Rebinto Sam

PART – 1:
UNICAST MESSAGE:

- The goal of PART 1 was that each process can send a unicast message to another process, to Update the vector clocks of the processes involved and print their vector clocks before and after sending/receiving the messages.
- Several libraries, including sys, threading, socket, time, random, and pickle, were used for various functionalities.
- With this software, there were a predetermined number of three processes.
- unicastVector.py asks the user for a distinct port number and a process ID, which can only be one of the numbers 1, 2, or 3.
- To save and display the vector of the particular process at each event, the software uses the EventList list.
- The vector_max(vector1, vector2) function calculates the vector using the formula: vector = [max(value) for value in zip(vector1, vector2)].
- Upon accepting the connection, the handler(conn, add) method prints the vector both before and after sending or receiving.
- Several threads are connected through a listen(node) function, which also receives messages.
- After entering the port number and process ID, we can communicate by sending a message to another process.
- The software collects port number of receiver, sender event number, receiver event number and sends the message with the help of these details.

```
import sys
import threading
import socket
import time
import random
import pickle
```

```
EventList = {} #To store the events of the processes

def vector_max(vector1,vector2):
    #To get the maximum value
    vector = [max(value) for value in zip(vector1,vector2)]
    return vector

def handler(conn,add):
    # Check Flag and determine the vector clocks
    print(f"\n[+] {add} is connected")
    received = conn.recv(1024)
    if received:
        print(f"Vector clock for all events before receiving:{EventList}\n")
        data = pickle.loads(received)
        rEventId = data["rEventId"]

        #updating vector value
        vector = vector_max(data["sEventData"] , EventList[rEventId])
        EventList[data["rEventId"]] = vector
        print(f"New Vector value for Event:{rEventId} is {vector} \n")
        received = None
    conn.close()

def listen(node):
    while True:
        conn,addr = node.accept()
        print(f"\nReceiving message from:{addr} \n")
        if conn and addr:
            thread = threading.Thread(target=handler,args=(conn,addr))
            thread.start()



def sender():
    while True:
        option = int(input("1. Enter 1 to Communicate\n2. Enter 2 to Skip\n3.
Enter 3 to Exit\n Enter your choice:"))
        print("\n")
        if option == 1:
            print(f"Vector clock for all events before sending:{EventList}\n")
            recieverPort = int(input("Enter port number of receiver: "))
            print("\n")
            if recieverPort:
                messagedata = {}

                sEventId = int(input("Enter Sender Event Number:"))
                print("\n")
                sEventData = EventList[sEventId]
                rEventId = int(input("Enter Reciever Event Number:"))
```

```
                print("\n")

                messagedata["sEventId"] = sEventId
                messagedata["sEventData"] = sEventData
                messagedata["rEventId"] = rEventId
                data = pickle.dumps(messagedata)
                try:
                    print(f"\nSending Message to 127.0.0.1:{recieverPort}\n")
                    conn = socket.socket()
                    conn.connect(('localhost',recieverPort))
                    print(f"\n[+] Connected and sending\n")

                    conn.sendall(data)
                    time.sleep(5)
                    conn.close()
                    print(f"Vector clock for all events after
sending:{EventList}\n")
                except Exception as e:
                    print(e)
                finally:
                    recieverPort = None
        elif option == 2:
            print("\n")
        else:
            print("Current Vector Clock for all Events\n")
            print(EventList)
            sys.exit()


def main(port):
    node = socket.socket()
    node.bind(('',port))

    # Start listening
    node.listen(10)

    listener_thread = threading.Thread(target=listen,args=(node,))
    sender_thread = threading.Thread(target=sender,args=())
    listener_thread.start()
    sender_thread.start()




if __name__ == '__main__':
    port = int(input("Enter port number for the node:"))
    print("\n")
    pId = int(input("Process ID for this node(1/2/3):"))
    print("\n")
```

```
    n1 = int(input(f"Enter the no. of events in Process {pId} : "))
    print("\n")
    e1 = [i for i in range(1, n1 + 1)]
    if pId == 1:
        EventList = {key: [key, 0, 0] for key in e1}
    elif pId == 2:
        EventList = {key: [0, key, 0] for key in e1}
    elif pId == 3:
        EventList = {key: [0, 0, key] for key in e1}
    print(EventList)
    print("\n")

    main(port)
```

OUTPUT:

```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>C:/Python311/python.exe "d:/Spring 23/Distributed Systems/Vignesh1002012757_Shane1002080770/Part 1/unicastVector.py"
Enter port number for the node:8000


Process ID for this node(1/2/3):1


Enter the no. of events in Process 1 : 3


{1: [1, 0, 0], 2: [2, 0, 0], 3: [3, 0, 0]}


New Vector value for Event:1 is [3, 0, 0]

Vector clock for all events after sending:{1: [3, 0, 0], 2: [2, 0, 0], 3: [3, 0, 0]}

1. Enter 1 to Communicate
2. Enter 2 to Skip
3. Enter 3 to Exit
 Enter your choice:3


Current Vector Clock for all Events

{1: [3, 0, 0], 2: [2, 0, 0], 3: [3, 0, 0]}
```

PART-2:
BROADCAST MESSAGE:

- The goal of PART 2 was that each process can broadcast a message to all other processes and to update the vector clocks of all the processes and print the vectors of all the processes before and after sending/receiving the messages.
- The working is pretty much similar to what was done in part 1. But the difference is that instead of the timestamp of the receiver getting updated, the timestamps of all processes get updated.
- Several libraries, including sys, threading, socket, time, random, and pickle, were used for various functionalities.
- With this software, there were a predetermined number of three processes.
- unicastVector.py asks the user for a distinct port number and a process ID, which can only be one of the numbers 1, 2, or 3.
- To save and display the vector of the particular process at each event, the software uses the EventList list.

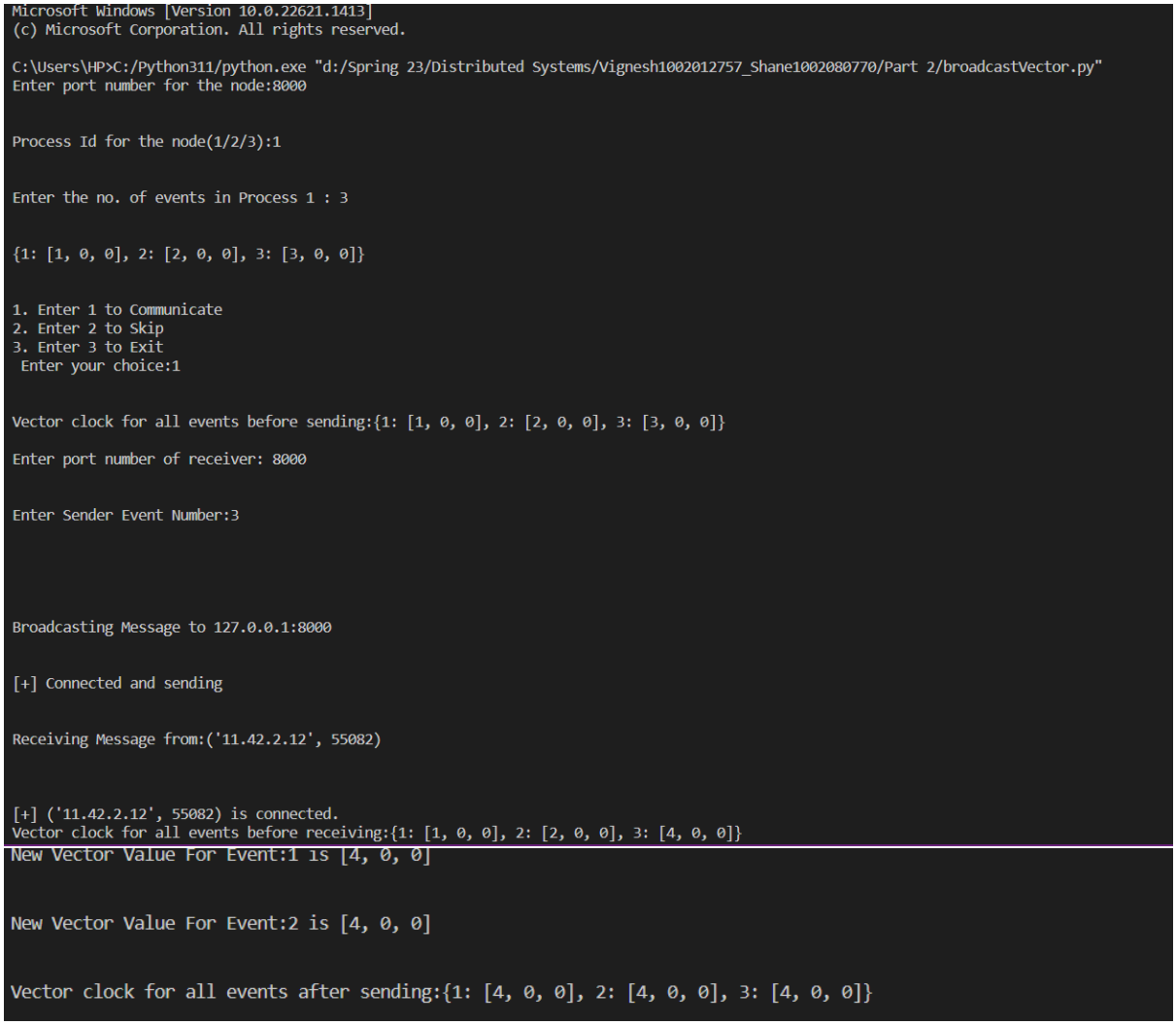
- The vector_max(vector1, vector2) function calculates the vector using the formula: vector = [max(value) for value in zip(vector1, vector2)].
- Upon accepting the connection, the handler(conn, add) method prints the vector both before and after sending or receiving.
- Several threads are connected through a listen(node) function, which also receives messages.
- After entering the port number and process ID, we can communicate by sending the message to all processes (broadcasting).
- The software collects port number of receiver, sender event number, receiver event number and broadcasts the message with the help of these details.

```
import sys
import threading
import socket
import time
import random
import pickle

EventList = {}
proId = 0

def vector_max(vector1,vector2):
    #To get the maximum value
    vector = [max(value) for value in zip(vector1,vector2)]
    return vector

def handler(conn,add):
    # Check Flag and determine vector clocks
    print(f"\n[+] {add} is connected.")
    received = conn
    if received:
        print(f"Vector clock for all events before receiving:{EventList}\n")
        data = pickle.loads(received)
        print("/n")

        for rEventId in EventList:
            if rEventId != data["sEventId"]:
                vector = vector_max(data["sEventData"] , EventList[rEventId])
                EventList[rEventId] = vector
                print(f"New Vector Value For Event:{rEventId} is {vector}")
                print("\n")

                if (rEventId + 1) in EventList:
                    for i in range(rEventId + 1, len(EventList) + 1):
                        EventList[i] = vector_max(EventList[i-1],EventList[i])

        received = None

def listen(node):
```

```
    while True:
        conn,addr = node.recvfrom(1024)
        print(f"\nReceiving Message from:{addr}")
        print("\n")
        if conn and addr:
            thread = threading.Thread(target=handler,args=(conn,addr))
            thread.start()



def sender():
    while True:
        option = int(input("1. Enter 1 to Communicate\n2. Enter 2 to Skip\n3.
Enter 3 to Exit\n Enter your choice:"))
        print("\n")
        if option == 1:
            print(f"Vector clock for all events before sending:{EventList}\n")
            recieverPort = int(input("Enter port number of receiver: "))
            print("\n")
            if recieverPort:
                messagedata = {}

                sEventId = int(input("Enter Sender Event Number:"))
                print("\n")
                EventList[sEventId][proId-1] += 1
                sEventData = EventList[sEventId]
                print("\n")

                messagedata["sEventId"] = sEventId
                messagedata["sEventData"] = sEventData
                data = pickle.dumps(messagedata)
                try:
                    print(f"\nBroadcasting Message to
127.0.0.1:{recieverPort}\n")
                    conn = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM,socket.IPPROTO_UDP)
                    conn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                    conn.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
#setting to broadcast mode
                    conn.connect(('localhost',recieverPort))
                    broadcast_address = ('<broadcast>', recieverPort)
                    print(f"\n[+] Connected and sending\n")

                    conn.sendto(data,broadcast_address)
                    time.sleep(5)
                    conn.close()
                    print(f"Vector clock for all events after
sending:{EventList}\n")
                except Exception as e:
```

```
                    print(e)
                finally:
                    recieverPort = None
        elif option == 2:
            print("\n")
        else:
            print("Current Vector Clock for all Events\n")
            print(EventList)
            sys.exit()


def main(port):
    node = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    node.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    node.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1) #setting to
broadcast mode

    node.bind(('',port))

    listener_thread = threading.Thread(target=listen,args=(node,))
    sender_thread = threading.Thread(target=sender,args=())
    listener_thread.start()
    sender_thread.start()




if __name__ == '__main__':
    port = int(input("Enter port number for the node:"))
    print("\n")
    pId = int(input("Process Id for the node(1/2/3):"))
    print("\n")
    proId = pId

    n1 = int(input(f"Enter the no. of events in Process {pId} : "))
    print("\n")
    e1 = [i for i in range(1, n1 + 1)]
    if pId == 1:
        EventList = {key: [key, 0, 0] for key in e1}
    elif pId == 2:
        EventList = {key: [0, key, 0] for key in e1}
    elif pId == 3:
        EventList = {key: [0, 0, key] for key in e1}
    print(EventList)
    print("\n")

    main(port)
```

OUTPUT:



WHAT I LEARNED:

- I learned how to implement a vector clock for a distributed system using socket programming.
- I even learned about the different libraries used for various functionalities, including sys, threading, socket, time, random, and pickle.
- I then learned about the use of EventList to save and display the vector of the particular process at each event, the handler function to accept connections and print vectors before and after sending/receiving messages, and the listener function to connect multiple threads and receive messages.

ISSUES ENCOUNTERED:

We couldn't figure out how to send one node's address to another, so we included an input statement that asked the other node's port number on which it wanted to communicate as a workaround.