CSE 5306 - DISTRIBUTED SYSTEMS

PROJECT 2

REPORT

REMOTE PROCEDURE CALL BASED COMMUNICATION

Note: I have neither given nor received unauthorized assistance on this work

Signed: ALETI SHIVANI REDDY **Signed: Chinmay Ramdas Hegde**
Date: 08-04-2022

NAME: Chinmay Ramdas Hegde

E-Mail ID: cxh9324@mavs.uta.edu

NetID: CXH9324

ID: 1002029323

NAME: SHIVANI REDDY ALETI

E-Mail ID: sxa4487@mavs.uta.edu

NetID: sxa4487

ID:1002034487

OVERALL STATUS

When assigned to this project, we had very basic knowledge of how to implement clocks. We took a glance at the project, then understood the problem statements and started working on it. Initially, we face few challenges to write code and get the files uploaded. But for that, we have gone through some websites, books and got a better idea of what communication protocols.

We were able to write the code in Java and deploy protocols to the server. Vector clocks implementation has been the biggest task for us in the project. We tried our best to learn and develop the best in java, but we got many errors initially. We finally developed the code for the given project and completed the task.

Who worked on part 1: Chinmay Ramdas Hegde

Who worked on part 2: Shivani Reddy Aleti

What issues were faced: The main challenge was handling interprocess communication without blocking the processes. Also, we have implemented the assignment so that it can be tested with N number of processes. At first understanding the concept of vector clocks was a little confusing which led to implementing the vector clock in a whole other new dimension. Later after we did a lot of research on it we had a clear view about the concept.

PART 1

Problem Statement: Berkeley's algorithm to synchronize these clocks to the average clock

What did we learn: We learned about Berkeley's algorithm and how to implement it using sockets.

What issues were faced: The main challenge was handling interprocess communication without blocking the processes. Also, we have implemented the assignment so that it can be tested with N number of processes.

How it was implemented: we used UDP socket-based communication for communicating among the nodes.

We created two files master(server) and slave(client).

Master node calculates the average difference between all the logical clock times received and the clock time given by the master's logical clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.

Below are the screen-shots of the running of the program

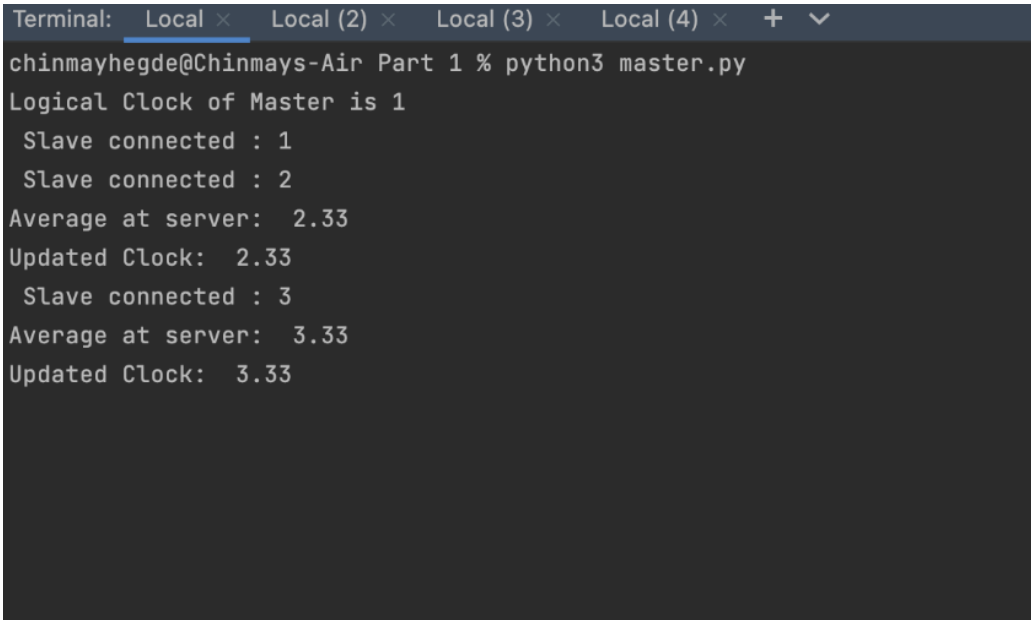


Fig 1: Terminal running Master displays the average clock time

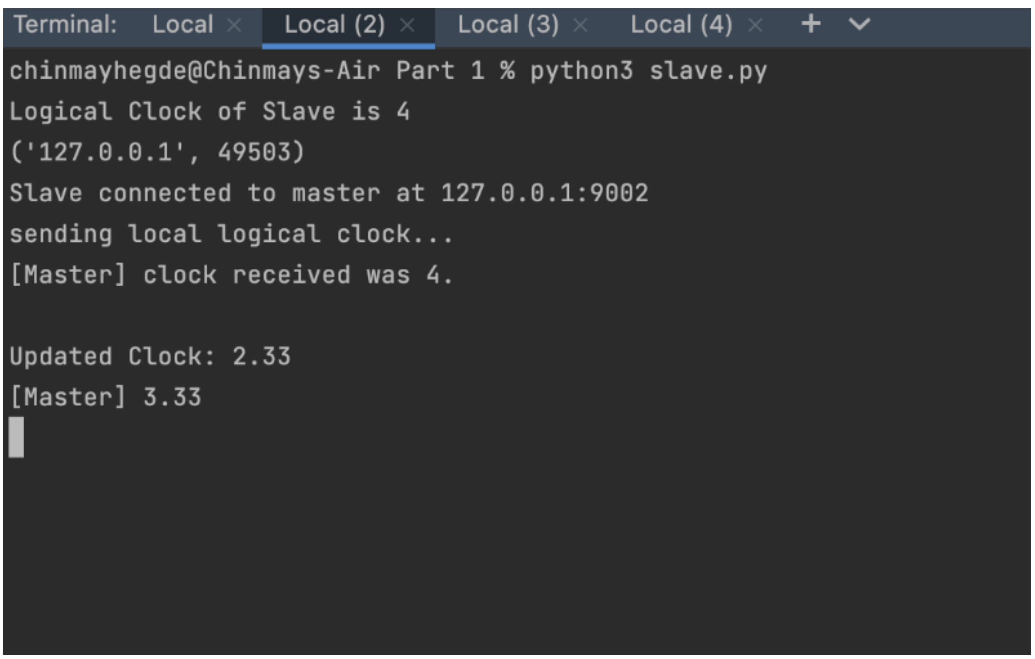


Fig 2: Terminal running Slave(1) displaying its own logical time and the average time after sync

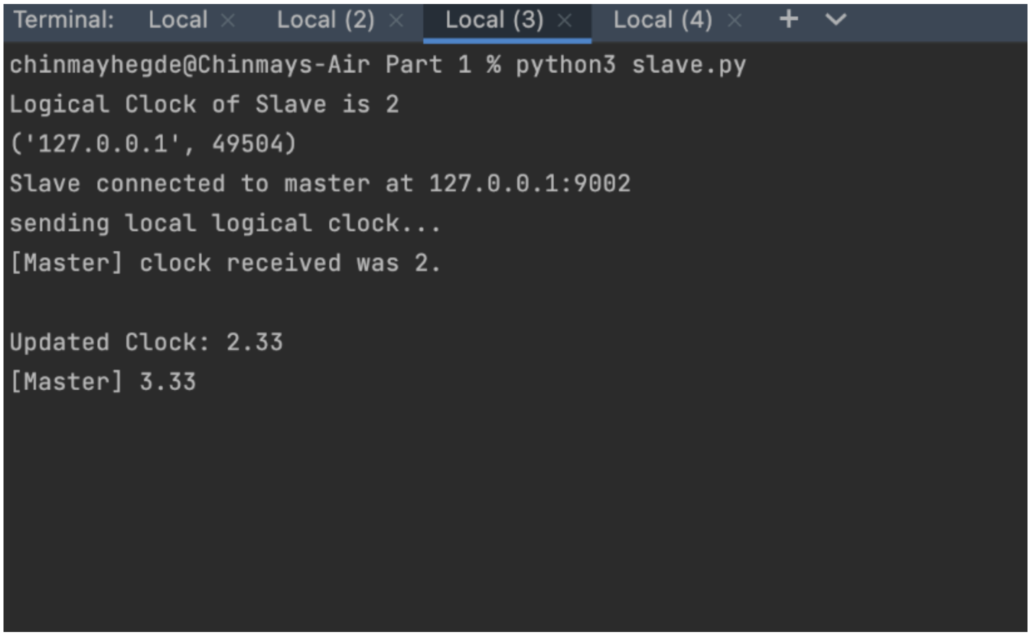


Fig 3: Terminal running Slave(2) displaying its own logical time and the average time after sync

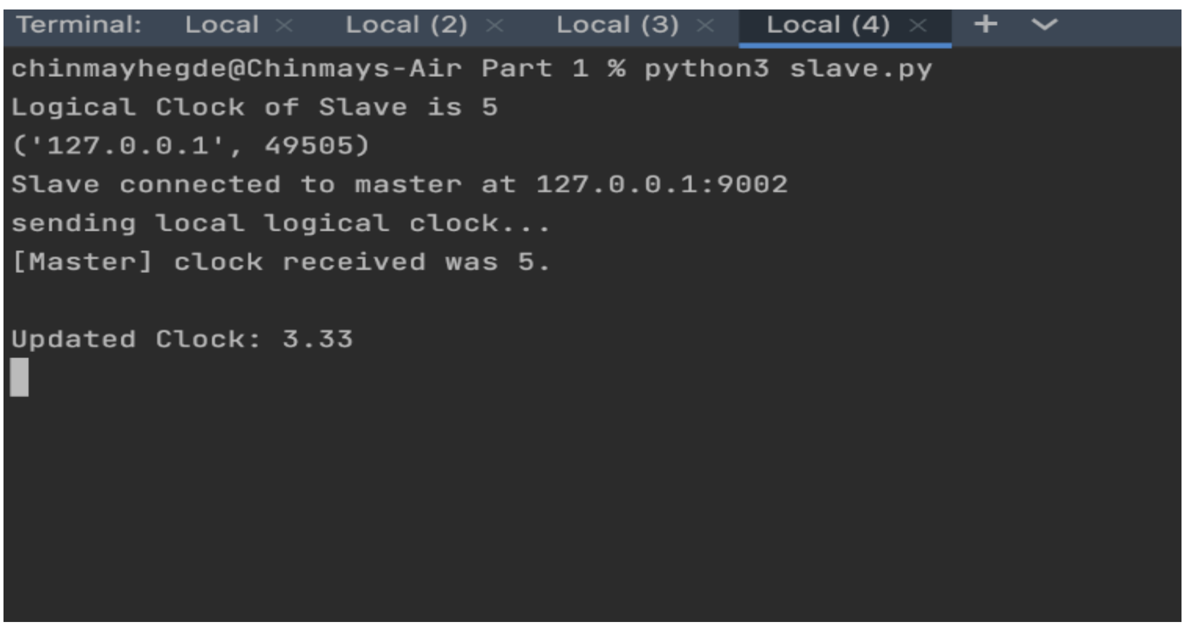


Fig 4: Terminal running Slave(3) displaying its own logical time and the average time after sync

PART 2

Problem statement: Implement the vector clock for your distributed system. You can create two threads for each process, one for sending messages to other nodes and one for listening to its communication port. Communication among nodes can be done using RPC or using sockets. Once a process sends a message, it should print its vector clock before and after sending the message. Similarly, once a process receives a message, it should print its vector clock before and after receiving the message. You can assume that the number of processes (machines) is fixed (equal to or larger than 3) and processes will not fail, join, or leave the distributed system.

What did we learn: We learnt How exactly the vector clock works using sockets.

References: we have taken references from github.
https://github.com/Devanshi1803/VectorClock-DistributedSystem.

Implementation: At first open the file client.py which is present in the folder part2.

- Open the terminal and navigate to the folder part2 in the terminal.
- Now try to run the given client.py folder by executing "**python client.py**" in terminal.

- Now open two more terminals and execute the same command "**python client.py**".

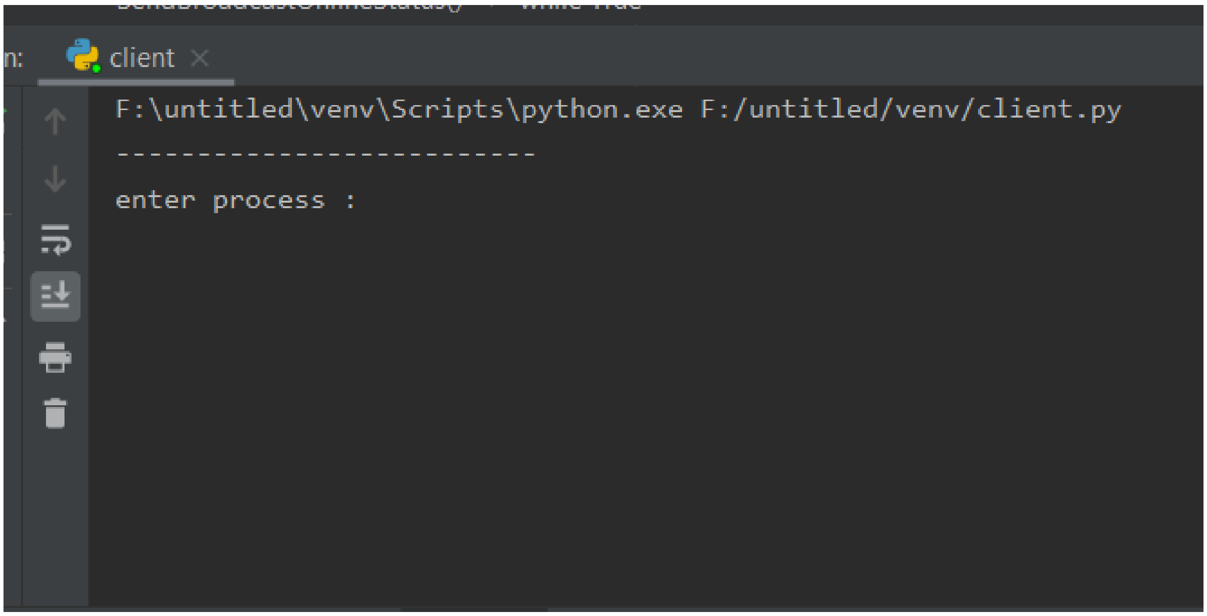


Fig 5: Executing the client.py in terminal one

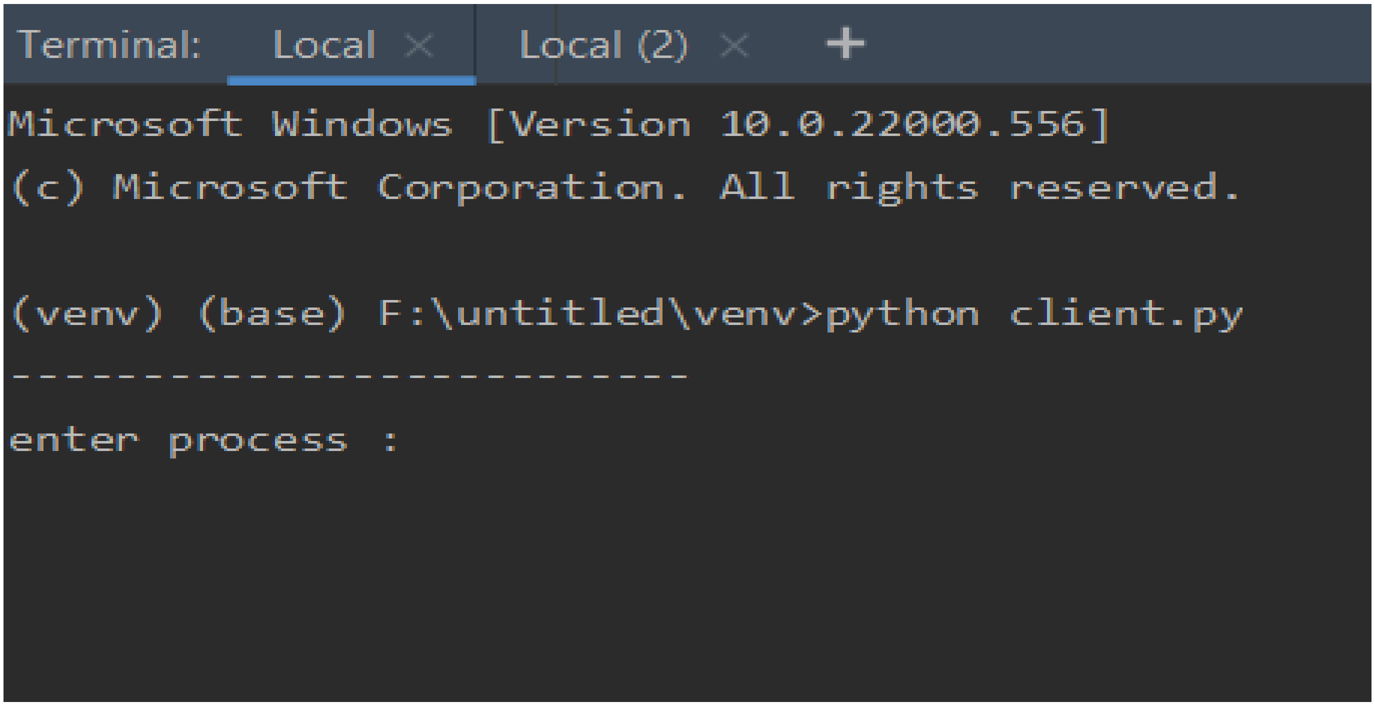


Fig 6: Executing the client.py in terminal two

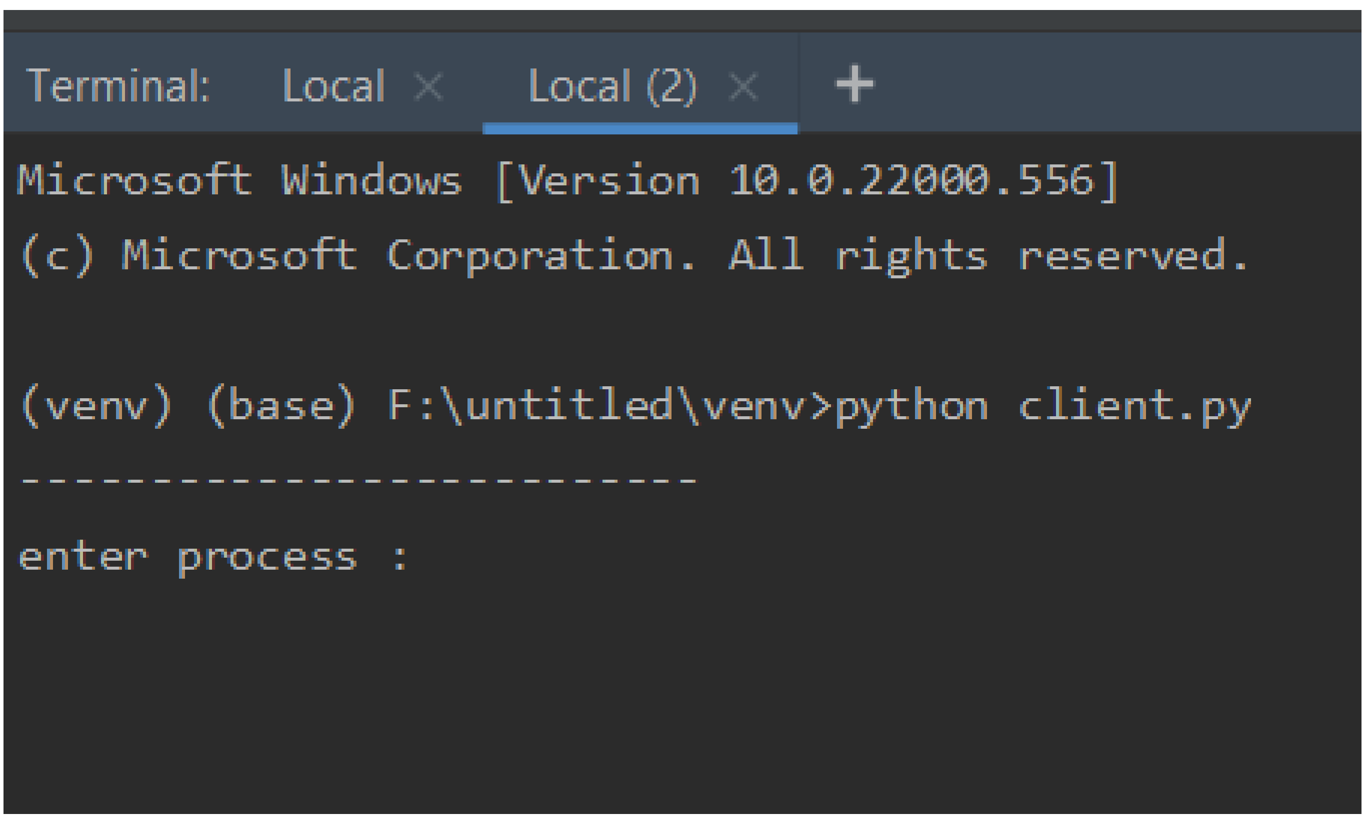
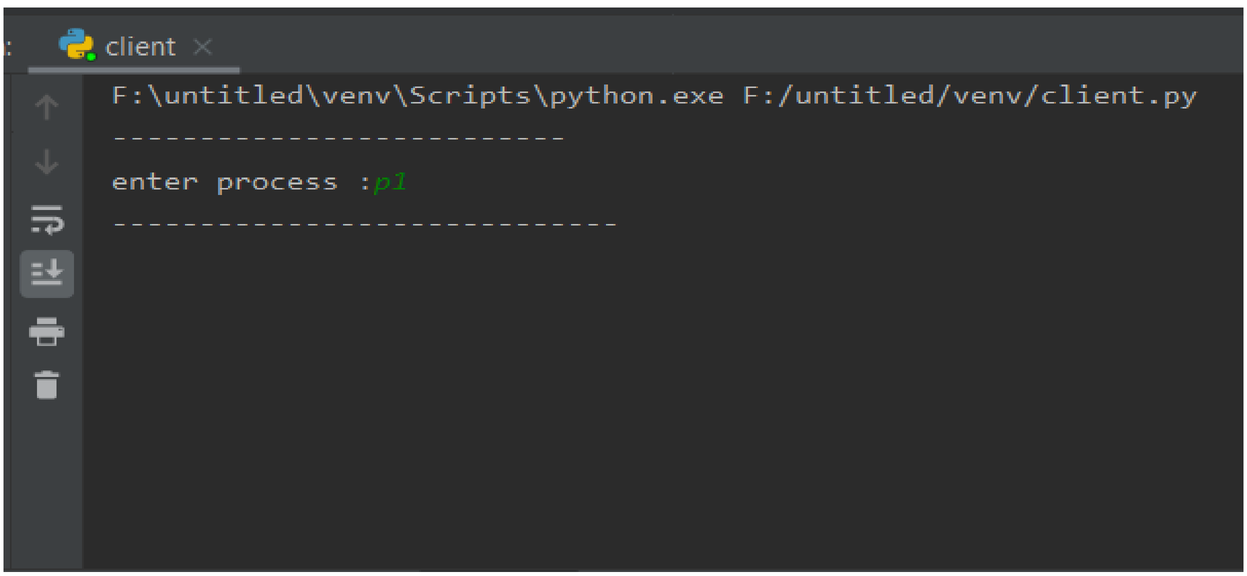
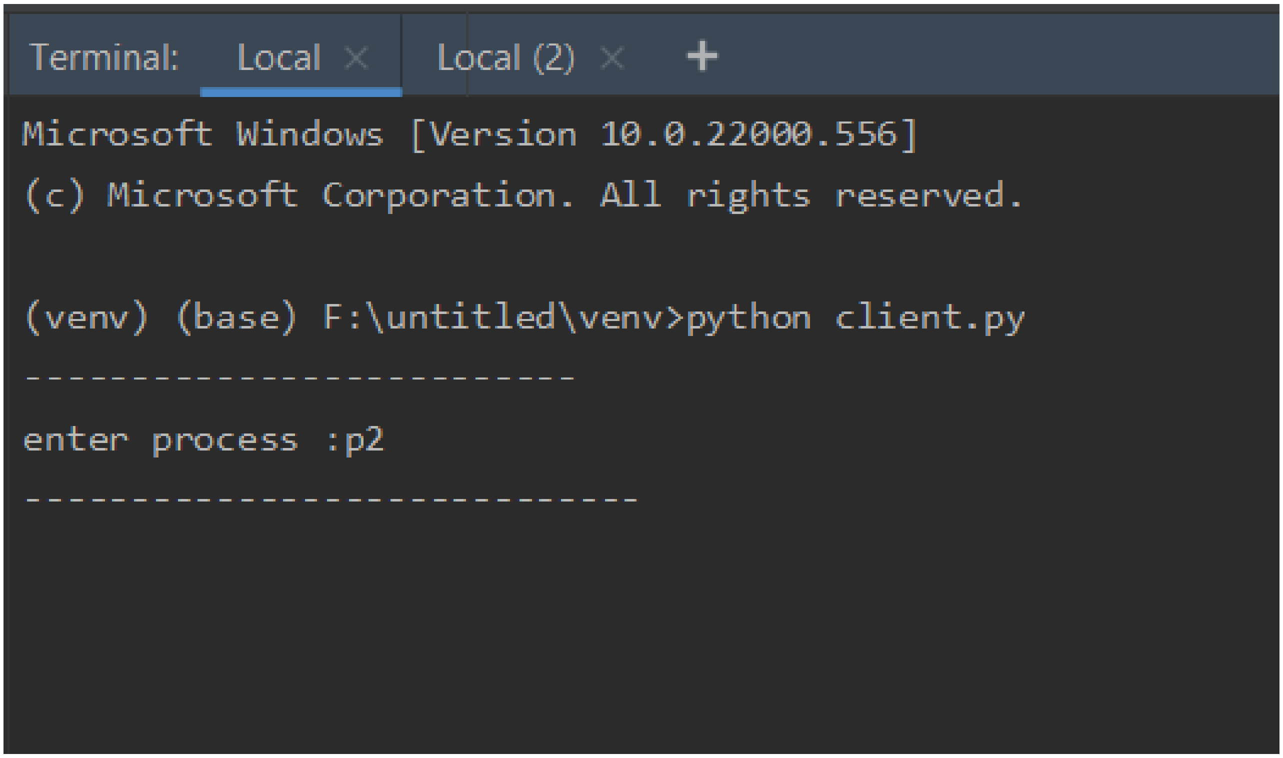


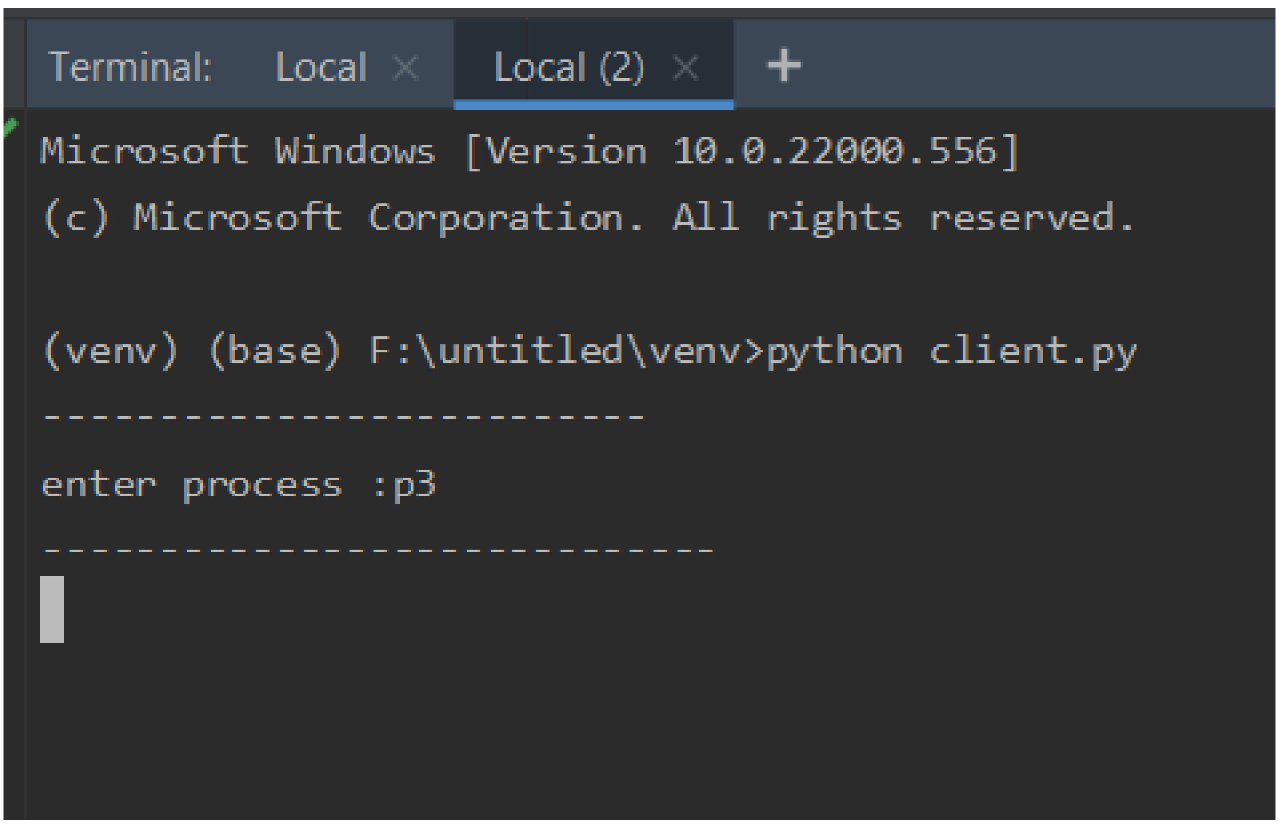
Fig 7: Executing the client.py in terminal three

The main reason behind executing client.py in three different terminals is to create three different process to create a communication between each node.(you can execute as many process as you want)

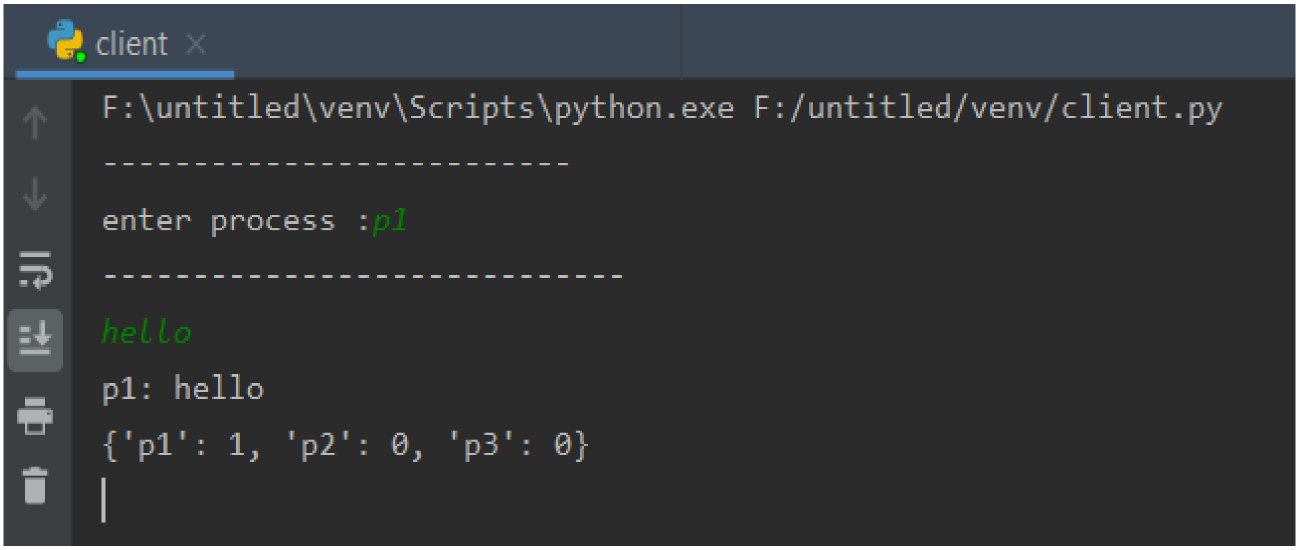
- Here comes the main part after you execute the given file in terminal one , Give a name to your process in terminal one (for eg: p1)
- Similar with cases terminal 2 (for eg: p2) and terminal 3(for eg: p3)




Terminal: Local Local (2) +
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.
(venv) (base) F:\untitled\venv>python client.py
----
enter process :p2


Terminal: Local Local (2) +
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.
(venv) (base) F:\untitled\venv>python client.py
----
enter process :p3

- Now p1 tries to send a message to rest of the peers in the network (for example: “ hello “)



- **{‘p1’: 1, ’p2’ : 0, ‘p3’ : 0}** states that p1 has sent the message to other peers, then the vector clock implements the vector p1 to 1 and the rest of the other two remains 0 as the process didn't send any message .
- Now let's check what is happening with the other two terminals.

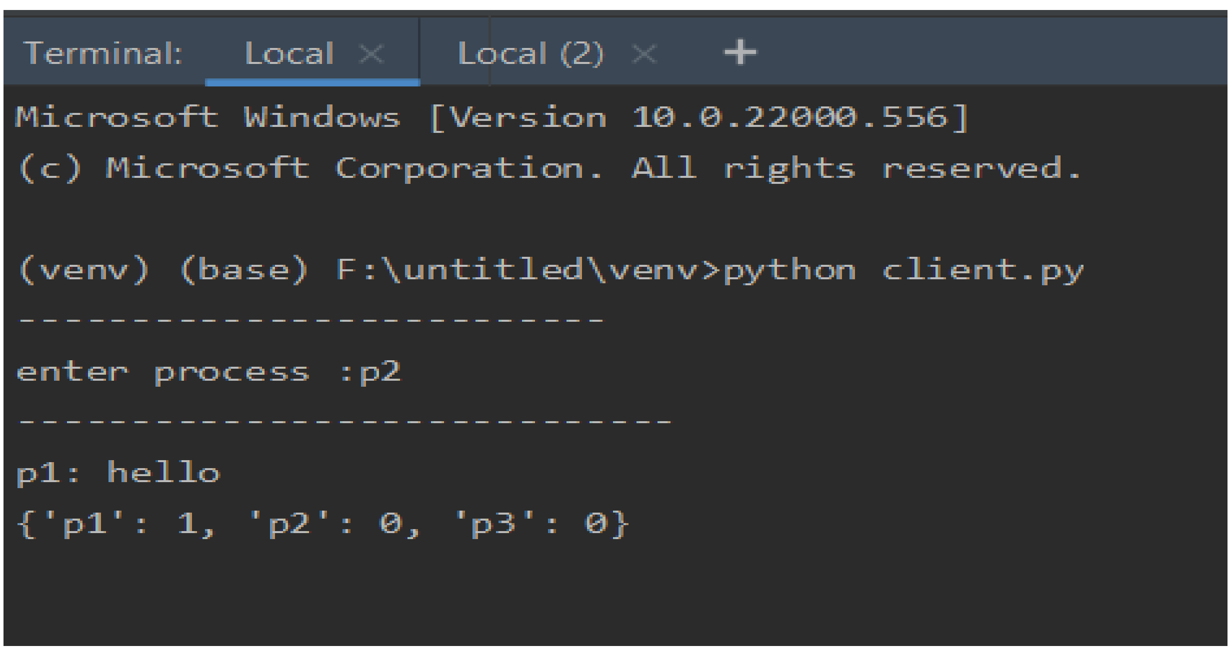


Fig 8: vector clock in terminal 2

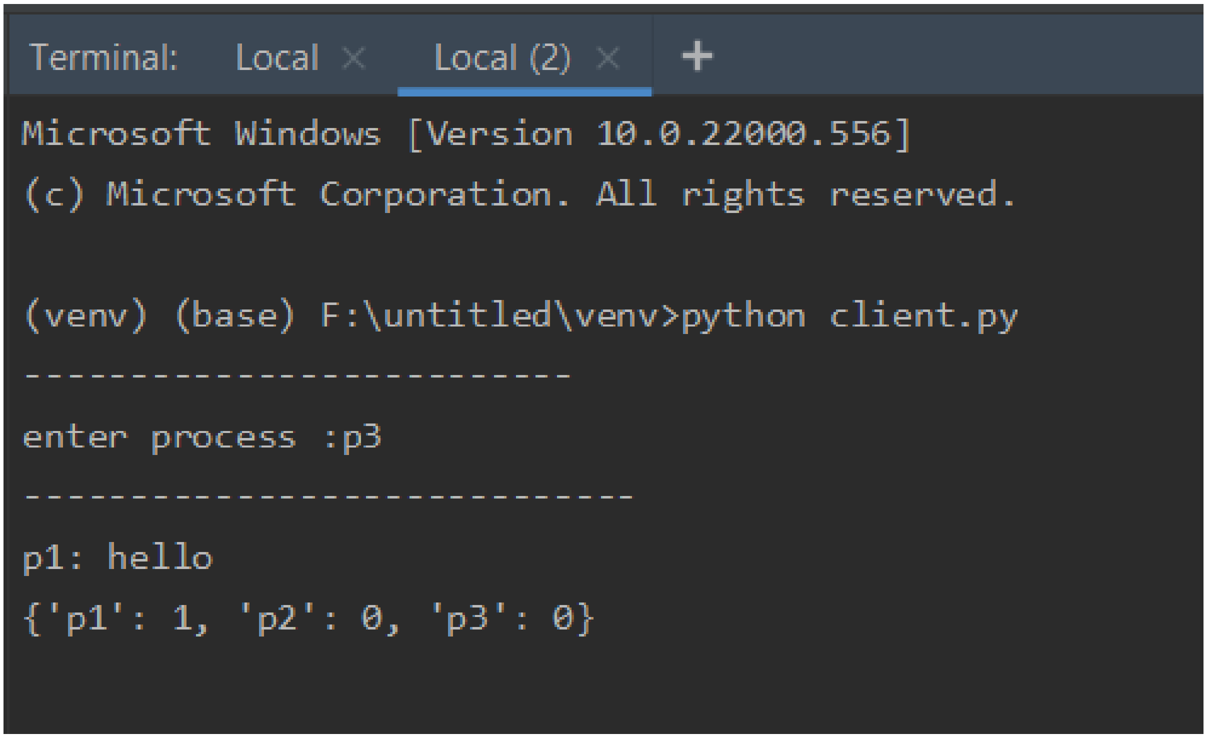
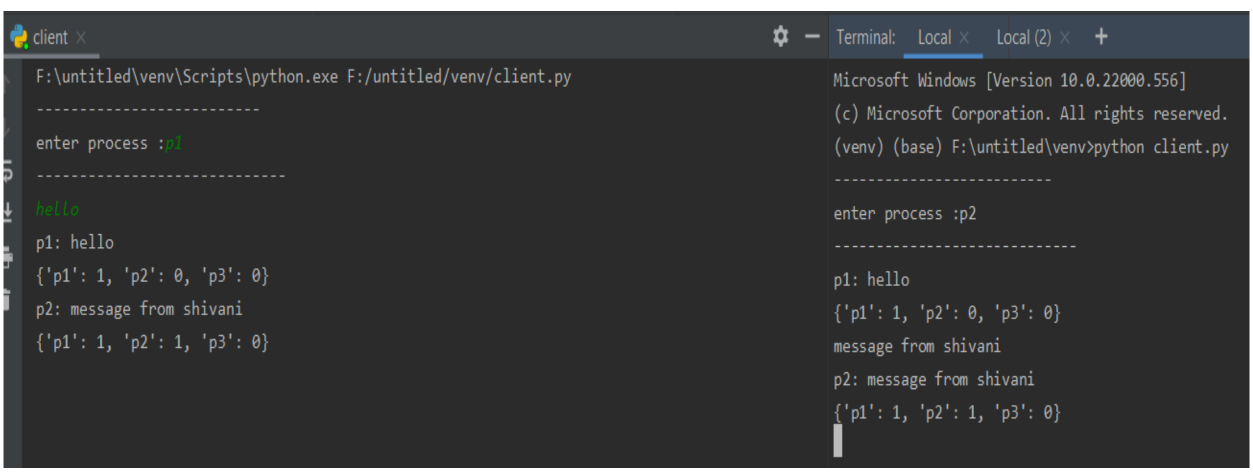


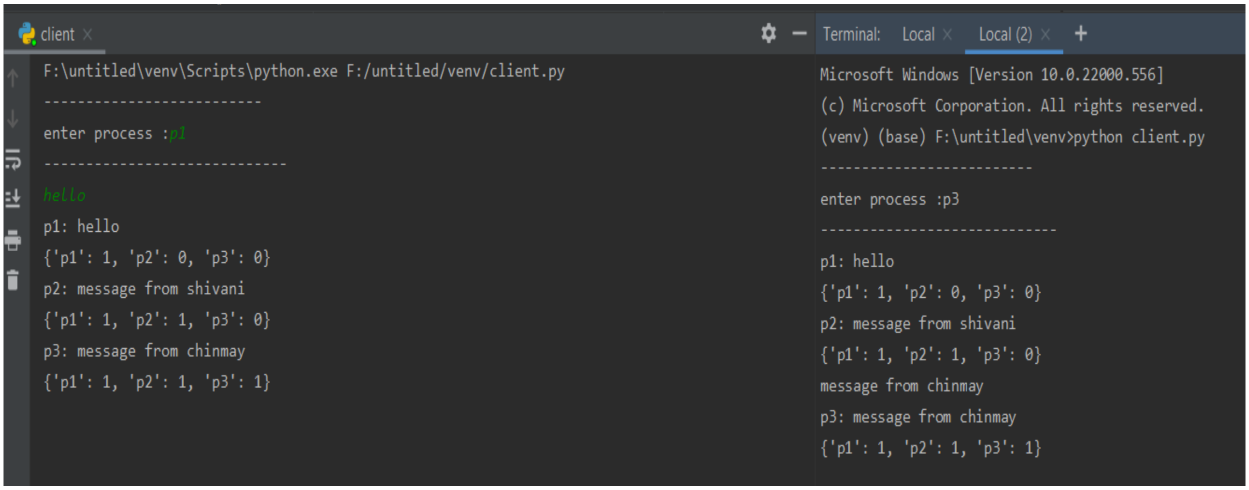
Fig 9: vector clock in terminal 3

Now let's try to send a message from p2 to other peers (for example: message from shivani).



If you observe the two terminals the value of p2 is getting incremented. Hence displays you the new vector clock.

Similarly sending message from p3 (for example: message from chinmay)



If you observe the two terminals the value of p3 is getting incremented. Hence displays you the new vector clock.

Hence you can keep on sending messages from different process and try to communicate with other peers. The vector clocks keeps on getting updates as per the communication.